An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range ⋆

Boris Houska, Hans Joachim Ferreau, Moritz Diehl

Eletrical Engineering Department, K.U. Leuven, Kasteelpark Arenberg 10 (bus 2446), 3001 Leuven, Belgium

Abstract

In this paper we present an automatic C-code generation strategy for real-time nonlinear model predictive control (NMPC), which is designed for applications with kilohertz sample rates. The corresponding code export module has been implemented within the software package `ACADO Toolkit`. It is capable of exporting fixed step-size integrators together with their sensitivities as well as a real-time Gauss-Newton method. Here, we employ the symbolic representation of optimal control problems in `ACADO` in order to auto-generate plain C-code which is optimized for final production. The exported code has been tested for model predictive control scenarios comprising constrained nonlinear dynamic systems with four states and a control horizon of ten samples. The numerical simulations show a promising performance of the exported code being able to provide feedback in much less than a millisecond.

Key words: Nonlinear Model Predictive Control, Automatic C-Code Generation, Real-Time Algorithms

1 Introduction

A recent trend in the field of convex optimization goes into the direction of automatic code export leading to automatically generated and customized interior point solvers. These optimized solvers have proven to be real-time feasible for online optimization with a sampling time in the microsecond range [22]. The advantages of auto-generated code are its efficiency and reliability. In addition, plain and self-contained C-code can easily be compiled for PC-like embedded hardware and possibly also for field-programmable gate arrays (FPGAs).

If a process model is derived from first-principle physical laws, we often end up with a nonlinear dynamic system, for which convex optimization techniques can typically not be applied. For this situation, nonlinear model predictive control (NMPC) algorithms are a well-known tool [4,5,16]. The idea to use code generation for NMPC has been introduced by Ohtsuka in form of the tool AutoGenU [23]. Computation times of 1.5 ms per iteration have been reported for an experimental hovercraft setup [25]. Another approach is the advanced step NMPC controller [26] which, however, solves a full nonlinear program at each sampling instant. Yet a different approach is the nonlinear real-time iteration (RTI) scheme [11,13]. Like the previous approaches, it uses a similar continuation Newton-type framework for which nominal stability has been shown [15] but solves one QP at each iteration. This allows for multiple active set changes and thus ensures that the nonlinear MPC algorithm cannot perform worse than a linear MPC controller. An overview of existing algorithms for fast nonlinear MPC can be found in [14].

The RTI scheme has originally been developed for large scale chemical engineering applications. The aim of this paper is to demonstrate that NMPC algorithms based on the RTI scheme can be optimized and auto-generated efficiently aiming at sampling times in the milli- and micro-second range. In order to allow for these ultra-fast execution times, we reduce the algorithmic components of the nonlinear real-time iteration scheme [13] to the absolute minimum. This allows us to auto-generate optimized C-code based on a symbolic representation of optimal control problems, as implemented within the open-source software `ACADO Toolkit` [19].

In Section 2 we introduce the algorithmic core of the real-time iteration scheme, while Section 3 explains the newly developed `ACADO Code Generation` tool. In Section 4 we

⋆ This paper was not presented at any IFAC meeting. Corresponding author B. Houska. Tel. +32-16-320364.

Email addresses: `boris.houska@esat.kuleuven.be` (Boris Houska), `joachim.ferrreau@esat.kuleuven.be` (Hans Joachim Ferreau), `moritz.diehl@esat.kuleuven.be` (Moritz Diehl).

demonstrate the performance of the implemented tools. Finally, the paper is concluded in Section 5.

2 The Real-Time Iteration Algorithm for Nonlinear Optimal Control

Throughout this paper we are interested in nonlinear optimal control problems of the form

$$\begin{aligned} \min_{\xi(\cdot),\zeta(\cdot)} \quad & \int_0^T \left(\|\xi(\tau)\|_2^2 + \|\zeta(\tau)\|_2^2 \right) \mathrm{d}\tau \\ \text{s.t.} \quad & \dot{\xi}(t) \;=\; f(\,\xi(t),\,\zeta(t)\,) \\ & \xi(0) \;=\; \xi_0 \\ & \underline{z} \;\le\; \zeta(t) \;\le\; \overline{z} \quad \text{for all } t \in [0,T]\;. \end{aligned} \tag{1}$$

Here, $\xi : \mathbb{R} \to \mathbb{R}^n$ denotes the state, $\zeta : \mathbb{R} \to \mathbb{R}^m$ the control input, and $\underline{z}, \overline{z} \in \mathbb{R}^m$ the control bounds. The right-hand side function f can be non-linear in both states and controls, while the objective is a least-squares tracking term with $\|\cdot\|_2$ denoting the Euclidean norm. In the context of nonlinear MPC, $\xi_0 \in \mathbb{R}^n$ is the current state measurement.

A more general optimal control problem formulation would also include non-autonomous dynamics, time-varying tracking references, a weighting in the objective, a quadratic Mayer term penalizing $\xi(T)$, non-linear state and control constraints, zero terminal constraints, etc. The proposed algorithms and software implementations can deal with all these issues as illustrated within the examples in Section 4. However, we prefer to keep our presentation simple and work for the moment with the formulation (1) which can later be generalized.

Recall that direct methods for optimal control [5,11] proceed in two steps: first, we discretize the problem. And second, we solve a finite dimensional nonlinear program.

2.1 Discretization of the Optimal Control Problem

For the discretization of the nonlinear dynamics several options exist. Collocation methods [5] directly represent the states as polynomials with a finite number of coefficients. Alternatively, single- or multiple shooting discretization methods [6,21,11] can be employed where an integrator is used in order to simulate the dynamic system. In this paper, we concentrate on single- and multiple shooting techniques which use in the simplest case a piecewise constant control discretization

$$\zeta(t) \approx \sum_{i=1}^{N} z_i I_{[t_i,t_{i+1})}(t)\;,$$

where $I_{[a,b)}(t)$ is equal to 1 if $t \in [a,b)$ and equal to 0 otherwise. The time sequence $0 = t_1 < t_2 < \ldots <$

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Fig. 1. A suitable Butcher tableau for a Runge-Kutta integrator with order 4 for fixed, pre-optimized step-sizes.

$t_{N+1} = T$ can e.g. be equidistant. We define

$$z := \left(z_1^T, \ldots, z_N^T\right)^T \in \mathbb{R}^{n_z}$$

with $n_z := Nm$ to achieve a convenient notation.

Let us regard the solution $\xi(t_{i+1})$ (with $i \in \{1,\ldots,N\}$) of the differential equation

$$\forall \tau \in [t_i, t_{i+1}]: \;\; \dot{\xi}(\tau) \;=\; f(\xi(\tau), z_i) \;\text{ and }\; \xi(t_i) = x_i$$

as a function $\Xi_i(x_i, z_i) = \xi(t_{i+1})$ depending on the discrete control input z_i and on the multiple shooting node x_i which is the initial value for the i-th control interval. Here, the operator Ξ_i can numerically be evaluated by using an integrator. As a reasonable step-size choice can often be pre-optimized before run-time, we suggest to employ a standard Runge-Kutta method using a constant step-size once the integrator is running in online mode. Note that a Runge-Kutta integrator whose Butcher tableau contains many zero entries can be beneficial. For example, exploiting the four zero-entries of the Butcher tableau of order 4 shown in Figure 1 reduces the computational load of each integration step by about one third.

We discretize the continuous least-squares objective as

$$\int_0^T \left(\|\xi(\tau)\|_2^2 + \|\zeta(\tau)\|_2^2 \right) \mathrm{d}\tau \;\approx\; \|F(x,y,z)\|_2^2$$

where $F(x,y,z) := \left(y^T,\, x^T,\, z^T \right)^T$. Here, the vector $x := \left(x_1^T, \ldots, x_N^T\right)$ summarizes the multiple shooting nodes but the initial value $y := x_0$. Moreover, we denote the multiple-shooting residual as

$$G(x,y,z) := \begin{pmatrix} x_1 - \Xi(y, z_1) \\ x_2 - \Xi(x_1, z_2) \\ \vdots \\ x_N - \Xi(x_{N-1}, z_N) \end{pmatrix}. \tag{2}$$

Now, we can summarize the result of the multiple-

shooting discretization as

$$\begin{aligned} \min_{x,y,z} \quad & \|F(x,y,z)\|_2^2 \\ \text{s.t.} \quad & y \;=\; \xi_0 \\ & 0 \;=\; G(x,y,z) \\ & \underline{z} \;\le\; z \;\le\; \overline{z}\,. \end{aligned} \tag{3}$$

This large but sparse nonlinear program must be solved in real-time and for changing measurement inputs ξ_0. The next section explains this in more detail.

2.2 Real-Time Iteration Algorithm

In order to solve least-squares NLPs of the form (3), generalized Gauss-Newton methods, as originally proposed in [6], have turned out to perform very well in practice. An offline full-step version of this method starts from an initial guess (x^0, y^0, z^0) and generates iterates of the form $x^+ = x + \Delta x$, $y^+ = y + \Delta y$ and $z^+ = z + \Delta z$ where $(\Delta x, \Delta y, \Delta z)$ solves the convex QP

$$\begin{aligned} \min_{\Delta x,\Delta y,\Delta z} \quad & \| F + F_x\Delta x + F_y\Delta y + F_z\Delta z \|_2^2 \\ \text{s.t.} \quad & y + \Delta y \;=\; \xi_0 \\ & G + G_x\Delta x + G_y\Delta y + G_z\Delta z \;=\; 0 \\ & \underline{z} \le\; z + \Delta z \;\le\; \overline{z}\,. \end{aligned} \tag{4}$$

Here, we have introduced the following short hands:

$$F \;:=\; F(x,y,z)\,, \quad F_x \;:=\; \partial_x F(x,y,z) \quad \text{etc.}$$

Although the Gauss-Newton method converges in general only linearly [1] to a local minimizer (x^*, y^*, z^*) of the problem (3), it can perform very well in practice if either the least-squares residual is small or if the function G is only mildly non-linear [6].

In the context of model predictive control, the above method is separated into a preparation and a feedback step [13], as the current measurement ξ_0 might not yet be available when we start solving the problem (3). In the preparation step, we evaluate F and G, compute the associated sensitivities F_x, F_y, F_z and G_x,G_y,G_z and perform a condensing step without knowing ξ_0 yet. Here, condensing refers to the computation of

$$\begin{aligned} R_y \;:=\; F_y - F_x G_x^{-1} G_y\,,\; R_z \;:=\; F_z - F_x G_x^{-1} G_z\,, \\ \text{and} \qquad\qquad R \;:=\; F - F_x G_x^{-1} G\,. \end{aligned} \tag{5}$$

[1] We assume that F and G are differentiable with Lipschitz continuous Jacobians, while the reduced Jacobian R_z from equation (5) is assumed to have always full column-rank.

Real Time Iterations for Nonlinear MPC:

Initialization: Choose initial values for (x, y, z).

Repeat Online:

1) Evaluate F, G and $F_{x,y,z}, G_{x,y,z}$ at (x, y, z).
2) Perform the condensing, i.e. compute R_y, R_z, R.
3) Wait for the measurement ξ_0.
4) Compute Q of the initial value embedding (7).
5) Solve the QP (8).
6) Send first control z_1^+ immediately to the process.
7) Update $(x, y, z) \leftarrow (x^+, y^+, z^+)$ and shift the time.

Fig. 2. An illustration of the real-time iteration scheme.

This means that the sparse quadratic problem (4) is reduced to a smaller QP of the form

$$\begin{aligned} \min_{\Delta y,\Delta z} \quad & \| R_y\Delta y + R_z\Delta z + R \|_2^2 \\ \text{s.t.} \quad & y + \Delta y \;=\; \xi_0 \\ & \underline{z} \le\; z_i + \Delta z_i \;\le\; \overline{z}\,. \end{aligned} \tag{6}$$

Once the measurement ξ_0 is available, we perform an initial value embedding step, i.e. we compute the matrix vector product

$$Q := R_y\,(\xi_0 - y) + R \tag{7}$$

constructing a dense and convex QP of the form

$$\min_{\Delta z} \quad \| R_z\Delta z + Q \|_2^2 \quad \text{s.t.} \quad \underline{z} \le\; z_i + \Delta z_i \;\le\; \overline{z} \tag{8}$$

which has only the input sequence Δz as a remaining degree of freedom. Solving this small and dense QP with a suitable QP solver completes the feedback step. Once its solution z^+ is available, we apply the first control z_1^+ to the process, shift the time horizon of the MPC, and perform the next preparation step. Figure 2 illustrates this real-time iteration idea in form of a pseudo code. For a mathematical foundation of this method including stability theorems, we refer to [3,11,15,26]. Please note that the above algorithm can also be transferred to the single shooting discretization if we use $x_{i+1} := \Xi(x_i, z_i)$ for all $i \in \{0, \ldots, N\}$. In this case we have always $G(x, y, z) = 0$ during the iteration, which implies $R \;=\; F$.

2.3 Limitations of the Real-Time Iteration Scheme

When using the above nonlinear MPC algorithm in real-world applications, the following issues may lead to a failure of the algorithm:

Infeasibility: If formulation (1) additionally comprises state constraints or non-convex control constraints, two types of infeasibility can occur: first, the

nonlinear online optimization problem itself can be infeasible. And second, the underlying quadratic programming problem within the SQP-type algorithm can become infeasible.

The first type of infeasibility is a general problem of NMPC algorithms. A possible remedy is the use of zero-terminal constraints and to solve the online optimization problem in every step exactly. Assuming that the first optimization problem including the zero-terminal constraint is feasible, that no uncertainties occur, and that exact state measurements are available, it can be shown that all online optimization problems remain feasible [24].

For handling the second type of infeasibility suitable strategies exist [10]. Note that QP infeasibility cannot occur if only control bounds $\underline{z} \leq z \leq \overline{z}$ are present.

Instability: Even if we assume that the online optimization problems all remain feasible, there are two reasons why the closed-loop system can become unstable. First, the closed-loop system can be unstable, as we have a finite horizon only. This problem can be addressed by using suitable end weights or zero terminal constraints [24]. Second, we might leave the region of contraction of the Gauss-Newton method, e.g. due to a large disturbance. This must be avoided employing suitable globalization strategies if necessary. Note that for the above real-time iteration scheme, local asymptotic closed-loop stability can be guaranteed assuming suitable end weights in combination with other regularity conditions [11,12,15].

3 The ACADO Code Generation Tool

The `ACADO Toolkit` is an open-source software tool for automatic control and dynamic optimization [19]. The aim of this section is to explain the newly developed `ACADO Code Generation` tool which makes use of the symbolic features of `ACADO` to export optimized C-code. Here, we follow an idea from [22], where automatic generation of C-code for convex optimization was suggested, and extend it to nonlinear dynamic systems.

3.1 Symbolic Representations of MPC Formulations

Let us consider a simple example for a nonlinear function defined as

$$f(\phi, \omega) \;:=\; -g\sin(\phi) - a\cos(\phi) - b\omega\,. \qquad (9)$$

Once we implement this function in plain C, we can evaluate it for a given input. However, for example the fact that f is affine in ω can not explicitly be detected if f is given in form of a standard C-function. In order

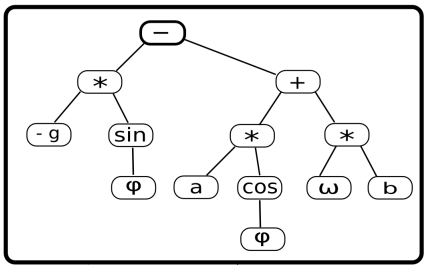

Fig. 3. The operator based tree-representation of the example function (9) as used in the software `ACADO Toolkit`.

```
#include <acado_toolkit.hpp>

int main( ){

// INTRODUCE THE VARIABLES:
// ------------------------
   DifferentialState     p;  // setup four
   DifferentialState     v;  // differential states
   DifferentialState   phi;
   DifferentialState omega;
   Control               a;  // setup control input

   DifferentialEquation  f;  // setup an ODE
   double T = 3.0;           // length of time horizon

   Matrix Q  = eye(4);       // weighting matrix Q
   Matrix R  = eye(1);       // weighting matrix R

// SETUP THE MPC FORMULATION:
// --------------------------
   OCP ocp( 0, T );          // construct an optimal
   ocp.minimizeLSQ( Q, R );  // control problem (OCP)
                             // with tracking objective

   f << dot(p)     == v;     // define four
   f << dot(v)     == a;     // ODE equations
   f << dot(phi)   == omega;
   f << dot(omega) == -g*sin(phi)-a*cos(phi)-b*omega;

   ocp.subjectTo( f );       // set model equations
   ocp.subjectTo( -1 <= a <= 1 );  // define bounds on
                                   // control input

// EXPORT TAILORED C-CODE:
// -----------------------
   MPCexport mpc(ocp);       // construct module for
   mpc.exportCode();         // auto-generating code

   return 0;
}
```

Fig. 4. A tutorial `C++` code for a MPC problem using `ACADO`.

to overcome this limitation, `ACADO Toolkit` represents functions in a symbolic form as illustrated in Figure 3.

This enables us for example to compute the derivative of f with machine precision using automatic differentiation or to detect the zero-entry in the Jacobian of f with the aim to generate highly efficient C-code.

In `ACADO` this concept of symbolic representation is employed to define the whole MPC optimization problem (cf. Figure 4). In this example, we define a least squares objective of the form

$$\int_0^T \left[\xi(\tau)^T \mathcal{Q} \xi(\tau) + \zeta(\tau)^T \mathcal{R} \zeta(\tau)\right] \mathrm{d}\tau\;,$$

where $\mathcal{Q}$ and $\mathcal{R}$ are not necessarily unit matrices. The

differential equation `f` in the tutorial code would in a mathematical notation be given by

$$\begin{aligned}\dot{p}(t) &= v(t)\\ \dot{v}(t) &= a(t)\\ \dot{\phi}(t) &= \omega(t)\\ \dot{\omega}(t) &= -g\sin(\phi(t)) - a(t)\cos(\phi(t)) - b\omega(t)\ ,\end{aligned} \tag{10}$$

where $\xi = (p, v, \phi, \omega)^T$ is the state and $\zeta = a$ the control. The control bounds have the form $-1 \leq a(t) \leq 1$.

For a more detailed documentation and for further tutorials on how to specify more general MPC formulations in `ACADO` we refer to [2,18].

3.2 Automatic Code Generation

Once a specific model predictive control problem has been set up with `ACADO`, we can export the code via the `MPCexport` class. This module will generate optimized C-code which is based on hard-coded dimensions and which uses static memory only. There are four major optimized C-functions generated:

- First, the possibly non-linear right-hand side as well as its derivatives with respect to the states and controls are exported as C-code. Here, the derivatives are symbolically simplified employing automatic differentiation tools and using zero-entries in the Jacobian.
- Second, a tailored Runge-Kutta method for the model equations is generated. This Runge-Kutta routine also integrates the associated variational differential equations which are needed to compute the derivatives of the function G. For non-adaptive step-sizes this is equivalent to automatic differentiation in forward mode.
- Third, a discretization algorithm is exported which organizes the single- or multiple-shooting evaluation together with the required linear algebra routines for condensing.
- Fourth, the real-time iteration Gauss-Newton method is auto-generated. At this point, the `ACADO Code Generation` tool employs a tailored algorithm for solving dense QPs of the form (8): either the code generation tool `CVXGEN` [22] to export a tailored C-code or an adapted variant of the online QP solver `qpOASES` [1] using fixed dimensions and static memory.

In order to illustrate how the exported code looks like, Figure 5 shows a snap-shot of an automatically generated initial value embedding step in plain C. We might still be able to guess that this piece of code implements a hard coded matrix-vector multiplication for a system with four states. However, auto-generated code is not designed to be easily readable but to be efficient and reliable.

```
[...]
void initialValueEmbedding(  ){
params.g[0] = acadoWorkspace.g[4] +
acadoWorkspace.H[4]*acadoWorkspace.deltaY[0] +
acadoWorkspace.H[18]*acadoWorkspace.deltaY[1] +
acadoWorkspace.H[32]*acadoWorkspace.deltaY[2] +
acadoWorkspace.H[46]*acadoWorkspace.deltaY[3];
params.g[1] = acadoWorkspace.g[5] +
acadoWorkspace.H[5]*acadoWorkspace.deltaY[0] +
[...]
```

Fig. 5. A snap-shot of automatically generated code: the produced C-code is hard to read but efficient and reliable.

Note that the `ACADO Code Generation` tool generates self-contained code, i.e. no additional libraries need to be linked. It does not contain any `if` or `switch` statements, i.e. we can completely exclude that the program runs into a part of code which we have accidentally never tested. Moreover, there are no `malloc/free` or `new/delete` statements in the auto-generated code. All the memory is static and global while the dimensions are hard-coded, i.e. no segmentation faults can occur. Moreover, we avoid `for`-loops whenever reasonable in order to ensure maximum efficiency, though this might also be done by the compiler.

Finally, the `ACADO Code Generation` tool offers an option to export code using single precision arithmetic. This is advantageous for certain hardware platforms but limits the applicability of the exported code to more well-conditioned problem formulations.

3.3 Remarks on Embedded QP Solvers

The `ACADO Code Generation` tool interfaces two QP solvers based on different algorithmic strategies:

The first one is a primal-dual interior-point solver which is auto-generated by the package `CVXGEN` [22]. The exported algorithm is implemented in highly efficient plain C code that only makes use of static memory. A major advantage of interior-point algorithms is their relatively constant calculation times for each occurring QP [8].

Active-set algorithms form a second class of suitable QP solvers, thus also the open-source package `qpOASES` [1] – which implements an online active set strategy [17] – is interfaced. For the `ACADO Code Generation` tool, a modification using hard-coded dimensions and static memory is employed. Calculation times of active-set solvers strongly depend on the number of required active set changes, which is hard to predict. On the other hand each active set iteration is much faster than an interior-point iteration. In addition, the availability of dedicated hot-starting procedures are an advantage of active set methods.

4 The Performance of the Auto-Generated NMPC Algorithm

In order to demonstrate the performance of auto generated NMPC algorithms we apply the `ACADO` autogeneration tools to two benchmark problems arising in mechatronics and chemical engineering. Both examples are tested using online optimization problem formulations with and without state constraints. Though we successfully employed both embedded QP solvers described in Section 3.3, only the run times of `qpOASES` are listed, which have been slightly lower throughout the examples.

4.1 NMPC for a crane model

Let us consider a crane with mass m, line length L, excitation angle ϕ, and horizontal trolley position p. Here, our control input is the acceleration a of the trolley. With v being the trolley velocity and ω being the angular velocity of the mass point, the system can be described by a simple but non-linear differential equation system of the form (10), where b is a positive damping constant. We use the parameters $m = 1\,\mathrm{kg}$, $L = 1\,\mathrm{m}$, $b = 0.2\,\mathrm{J\,s}$ as well as $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$.

Now, we export a nonlinear MPC code using the `ACADO Code Generation` tool. Our MPC formulation coincides with the problem (1), where $\xi := (p, v, \phi, \omega)^T$ is the state and $\zeta := a$ the control while the corresponding right-side function f is given above. The control bounds are $\underline{z} = -1$ and $\overline{z} = 1$. Addtionally, zero terminal constraints are imposed at the end of the horizon.

Running the real-time loop leads to a computation time of about $95\,\mu$s per real-time iteration (or about $86\,\mu$s without zero-terminal constraints). This result has been obtained on a 2.8 GHz processor with 4 GB RAM by running the real-time iteration loop 10^4 times and taking the average. Note that the compiled code for the whole controller has a size of 160 kB (under Linux) and requires 14 kB for storing problem data and intermediate results. Table 1 shows a more detailed list with the computation times. Here, we have employed a Runge-Kutta integrator of order 4 using 20 integrator steps which yields an sufficient integrator accuracy of $\approx 10^{-3}$. The time horizon of $T = 3\,\mathrm{s}$ was divided into 10 control intervals, which determines the dimension of the dense QP.

Table 1
Run-time performance of the auto-generated NMPC algorithm applied to the crane model.

	CPU time	%
Integration & sensitivities	53 μs	56 %
Condensing	24 μs	25 %
QP solution (with `qpOASES`)	13 μs	13 %
Remaining operations	< 5 μs	< 6 %
One complete real-time iteration	95 μs	100 %

Note that the time for the feedback step is mainly determined by the time which is needed to solve the dense QP (8). Due to the efficiency of the QP solution, the time between availability of the measurement and the application of the new control is only $13\,\mu$s. The code generated by `ACADO` allows us to perform the preparation step within the remaining $82\,\mu$s.

4.2 NMPC for a Continuous Stirred Tank Reactor

Second, we consider the benchmark problem of a continuous stirred tank reactor (CSTR) with four states and two controls. Here, the first two states, c_A and c_B, are the concentrations of cyclopentadiene (substance A) and cyclopentenol (substance B), respectively, while the other two states, ϑ and ϑ_K, denote the temperature in the reactor and temperature in the cooling jacket of the tank reactor. The state vector is $\xi = (\, c_A,\, c_B,\, \vartheta,\, \vartheta_K\,)^T$. Our first input is the feed inflow which is controlled via its scaled rate $\zeta_1 = \frac{\dot{V}}{V_R}$, while the temperature ϑ_K is held down by an external heat exchanger whose heat removal rate $\zeta_2 = \dot{Q}_K$ can be controlled as well.

The following nonlinear model can be found in [11,20]:

$$\begin{aligned}
\dot{c}_A(t) &= u_1(c_{A0} - c_A(t)) - k_1(\vartheta(t))c_A(t) \\
&\quad -k_3(\vartheta(t))(c_A(t))^2 \\
\dot{c}_B(t) &= -u_1 c_B(t) + k_1(\vartheta(t))c_A(t) - k_2(\vartheta(t))c_B(t) \\
\dot{\vartheta}(t) &= u_1(\vartheta_0 - \vartheta(t)) + \frac{k_w A_R}{\rho C_p V_R}(\vartheta_K(t) - \vartheta(t)) \\
&\quad -\frac{1}{\rho C_p}\left[\, k_1(\vartheta(t))c_A(t)H_1 + k_2(\vartheta(t))c_B(t)H_2\right. \\
&\quad \left. +k_3(\vartheta(t))(c_A(t))^2 H_3\,\right] \\
\dot{\vartheta}_K(t) &= \frac{1}{m_K C_{PK}}\left(u_2 + k_w A_R(\vartheta(t) - \vartheta_K(t))\right)\ .
\end{aligned}$$

We set up a closed-loop scenario using the above model, where the parameters, control bounds, and end weights are taken from [11]. For illustration, we chose three different set points, one of which is constructed such that it can not be tracked exactly due to over-restrictive state constraints:

$$\vartheta(t) \geq 98\,^\circ C\,,\ \vartheta_K(t) \geq 92\,^\circ C\,.$$

The first set point, simulated for $t \in [0, 3000\mathrm{s}]$ corresponds to the choice in [11]. Moreover, the closed-loop setup is similar to the one reported in [7,9] where the main difference is that we divide the prediction horizon of $T = 1500\,\mathrm{s}$ into 10 instead of 22 control intervals of equal length (which was found to hardly affect the control performance).

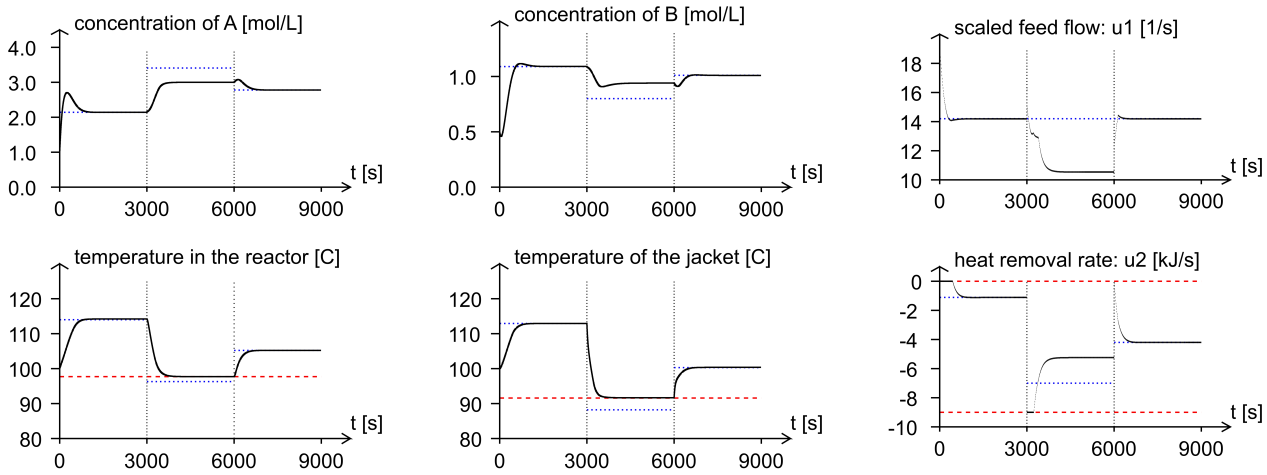

Fig. 6. Simulation of a closed-loop scenario for the continuous stirred tank reactor showing all four states and the two control inputs. The second set-point (dotted and blue) cannot be reached due to the over-restrictive constraints (dashed and red).

Figure 6 shows the result of the closed-loop simulation using auto-generated code (binary size of 220 kB and 25 kB for the data). We use a sampling time of 5 s and employ a Runge-Kutta integrator of order 4 using 20 integrator steps. Running it on the same hardware as used for the crane example, we obtain the run-times listed in Table 2. The worst measured total run-time for one real-time iteration is about 400 μs, which is several orders of magnitude faster than run-times reported earlier. For example, computation times in the order of minutes for a very similar setup have been reported fifteen years ago in [9], while [7,11] report computation times of about one second (considering the computer hardware used that time, not more than a factor of 10 in the run-time difference can be explained by the speed-up of PCs). If state constraints are left away, the QP solution time for our scenario reduces to less than 30 μs (and condensing becomes sligtly faster), thus an overall real-time iteration would take not more than 240 μs in that case [2].

Note that up to 68% (or 45% if no state constraints are present) of the computation time is spent in the QP solver and the condensing routine, whose computational load grows cubically with the number of control intervals. Thus, when longer control horizons are necessary, a tailored sparse QP solver should be used instead.

5 Conclusions

In this paper, we have discussed automatic code generation techniques for nonlinear model predictive control algorithms. We have reviewed the nonlinear real-time iteration scheme, which has originally been developed in [13] for large scale chemical processes, and showed how ultra-fast computation times for embedded applications can be achieved. The presented `ACADO Code Generation` tool exports optimized, self-contained C-code for the model simulation and sensitivity generation as well as for the Gauss-Newton algorithm.

Table 2
Worst-case run-time performance of the auto-generated NMPC algorithm applied to the CSTR model using 10 control intervals with state constraints.

	CPU time	%
Integration & sensitivities	121 μs	30 %
Condensing	98 μs	24 %
QP solution (with `qpOASES`) [3]	180 μs	44 %
Remaining operations	< 5 μs	< 2 %
A complete real-time iteration	404 μs	100 %

Moreover, the automatic code generation has been tested for two benchmark processes. For a simple but nonlinear crane model with four states, one control input, and ten control intervals, one real-time iteration takes 95 μs on a modern standard computer. For a more challenging chemical benchmark problem with two control inputs the worst measured computation time was found to be about 400 μs.

An important topic for future research will be to employ a sparse QP solver that can directly deal with the un-condensed, large-scale quadratic program. Moreover, running the auto-generated C-code on PC-like embedded hardware, or even field programmable gate arrays (FPGAs), might be an interesting research topic for engineers with nonlinear real-world applications. Further-

[2] Auto-generated code using 22 control intervals and no state constraints would still be very fast taking less than 1.5 ms per real-time iteration.

[3] This worst-case runtime corresponds to 21 active set changes required once during transition to the infeasibile set point. Each additional change would require 6.5 μs extra.

more, the concept of automatic code generation might also be transferred to integration and sensitivity generation for large scale dynamic systems.

Note that the `ACADO Code Generation` tool, which was developed along with this paper, is freely available under the LGPL license, see [2].

Acknowledgments

Research supported by Research Council KUL: CoE EF/05/006 Optimization in Engineering(OPTEC), OT/03/30, IOF-SCORE-S4CHEM, GOA/10/009 (MaNet), GOA/10/11, several PhD/-postdoc and fellow grants; Flemish Government: FWO: PhD/-postdoc grants, projects G.0452.04, G.0499.04, G.0211.05, G.0226.06, G.0321.06, G.0302.07, G.0320.08, G.0558.08, G.0557.08, G.0588.09,G.0377.09, research communities (ICCoS, ANMMM, MLDM); IWT: PhD Grants, Belgian Federal Science Policy Office: IUAP P6/04; EU: ERNSI; FP7-HDMPC, FP7-EMBOCON, Contract Research: AMINAL. Other: Helmholtz-viCERP, EMBOCON, COMET-ACCM. The second author holds a PhD fellowship of the Research Foundation – Flanders (FWO).

References

[1] `qpOASES` Homepage. http://www.qpOASES.org, 2007–2011.

[2] ACADO Toolkit Homepage. http://www.acadotoolkit.org, 2009–2011.

[3] F. Allgöwer, T.A. Badgwell, J.S. Qin, J.B. Rawlings, and S.J. Wright. Nonlinear Predictive Control and Moving Horizon Estimation – An Introductory Overview. In P. M. Frank, editor, *Advances in Control, Highlights of ECC'99*, pages 391–449. Springer, 1999.

[4] F. Allgöwer and A. Zheng. *Nonlinear Predictive Control*, volume 26 of *Progress in Systems Theory*. Birkhäuser, Basel Boston Berlin, 2000.

[5] L.T. Biegler and J.B Rawlings. Optimization approaches to nonlinear model predictive control. In W.H. Ray and Y. Arkun, editors, *Proc. 4th International Conference on Chemical Process Control - CPC IV*, pages 543–571. 1991.

[6] H.G. Bock. Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*. Birkhäuser, Boston, 1983.

[7] H.G. Bock, M. Diehl, D.B. Leineweber, and J.P. Schlöder. A direct multiple shooting method for real-time optimization of nonlinear DAE processes. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, volume 26 of *Progress in Systems Theory*, pages 246–267. Birkhäuser. Boston, 2000.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. University Press, Cambridge, 2004.

[9] H. Chen. *Stability and Robustness Considerations in Nonlinear Model Predictive Control*. Fortschr.-Ber. VDI Reihe 8 Nr. 674. VDI Verlag, Düsseldorf, 1997.

[10] A.R. Conn, N. Gould, and P.L. Toint. *Trust-Region Methods*. MPS/SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[11] M. Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*. PhD thesis, Universität Heidelberg, 2001.

[12] M. Diehl, H.G. Bock, and J.P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005.

[13] 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. *J. Proc. Contr.*, 12(4):577–585, 2002.

[14] M. Diehl, H. J. Ferreau, and N. Haverbeke. *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pages 391–417. Springer, 2009.

[15] M. Diehl, R. Findeisen, and F. Allgöwer. A Stabilizing Real-time Implementation of Nonlinear Model Predictive Control. In L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, editors, *Real-Time and Online PDE-Constrained Optimization*, pages 23–52. SIAM, 2007.

[16] M. Diehl, I. Uslu, R. Findeisen, S. Schwarzkopf, F. Allgöwer, H.G. Bock, T. Bürner, E.D. Gilles, A. Kienle, J.P. Schlöder, and E. Stein. Real-Time Optimization for Large Scale Processes: Nonlinear Model Predictive Control of a High Purity Distillation Column. In M. Grötschel, S. O. Krumke, and J. Rambau, editors, *Online Optimization of Large Scale Systems: State of the Art*, pages 363–384. Springer, 2001.

[17] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[18] H.J. Ferreau, B. Houska, T. Kraus, and M. Diehl. Numerical Methods for Embedded Optimisation and their Implementation within the ACADO Toolkit. In W. Mitkowski R. Tadeusiewicz, A. Ligeza and M. Szymkat, editors, *7th Conference - Computer Methods and Systems (CMS'09)*, Krakow, Poland, November 2009.

[19] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 2011. DOI: 10.1002/oca.939 (in print).

[20] K.-U. Klatt and S. Engell. Rührkesselreaktor mit Parallel- und Folgereaktion. In S. Engell, editor, *Nichtlineare Regelung – Methoden, Werkzeuge, Anwendungen. VDI-Berichte Nr. 1026*, pages 101–108. VDI-Verlag, Düsseldorf, 1993.

[21] D.B. Leineweber, 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. *Computers and Chemical Engineering*, 27:167–174, 2003.

[22] J. Mattingley and S. Boyd. *Automatic Code Generation for Real-Time Convex Optimization*. Convex Optimization in Signal Processing and Communications, Y. Eldar and D. Palomar, Eds., Cambridge University Press, 2009.

[23] T. Ohtsuka. A Continuation/GMRES Method for Fast Computation of Nonlinear Receding Horizon Control. *Automatica*, 40(4):563–574, 2004.

[24] J.B. Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill, 2009.

[25] H. Seguchi and T. Ohtsuka. Nonlinear Receding Horizon Control of an Underactuated Hovercraft. *International Journal of Robust and Nonlinear Control*, 13(3–4):381–398, 2003.

[26] V. M. Zavala and L.T. Biegler. The Advanced Step NMPC Controller: Optimality, Stability and Robustness. *Automatica*, 45:86–93, 2009.