Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 51-20 (2018) 374–380

Towards a modular software package for embedded optimization

Robin Verschueren * Gianluca Frison * Dimitris Kouzoupis *
Niels van Duijkeren ** Andrea Zanelli * Rien Quirynen ***
Moritz Diehl *

* *Department of Microsystems Engineering and Department of Mathematics, University of Freiburg, Germany (e-mail: robin.verschueren@imtek.uni-freiburg.de).*
** *Department of Mechanical Engineering, KU Leuven, Belgium.*
*** *Mitsubishi Electric Research Labs, Cambridge, MA, USA.*

Abstract: In this paper we present `acados`, a new software package for model predictive control. It provides a collection of embedded optimization algorithms written in `C`, with a strong focus on computational efficiency. Its modular structure makes it useful for rapid prototyping, i.e. designing a control algorithm by putting together different algorithmic components that are readily connected and interchanged. The usefulness of the software is demonstrated with a closed-loop simulation experiment of an inverted pendulum, which shows `acados` attaining sub-millisecond computation times per iteration. Furthermore, we showcase a new algorithmic idea in the context of embedded nonlinear model predictive control (NMPC), namely sequential convex quadratic programming (SCQP), along with an efficient implementation of it.

Keywords: Nonlinear Model Predictive Control, Embedded Optimization, Software

1. INTRODUCTION

In the past few decades, nonlinear model predictive control (NMPC) has been increasingly adopted for industrial applications of automatic control (Ferreau et al., 2017). Crucial for the use of advanced NMPC-based technologies is the availability of a sufficiently powerful software framework that provides a flexible, reliable and user-friendly implementation of computationally efficient NMPC algorithms (Diehl et al., 2009). More specifically, such a framework of state-of-the-art optimal control algorithms should be relatively easy to use by control practitioners, i.e., it should not only be directed towards experts in the field of numerical optimization.

In the context of real-time optimal control on embedded hardware, the technique of automatic code generation has experienced an increasing popularity over the past decade. Originally, code generation tools were developed for convex quadratic programming, such as in `CVXGEN` (Mattingley and Boyd, 2009), `FORCES` (Domahidi and Perez, 2013) and μAO-`MPC` (Zometa et al., 2013). However, several software packages have already been developed also for real-time nonlinear MPC, such as the `C/GMRES` solver (Ohtsuka and Kodama, 2002), the `ACADO Toolkit` and its code generation tool in (Houska et al., 2011) or the `VIATOC` software (Kalmari et al., 2015). More recently, `FORCES Pro` included the code generation of interior point methods for NMPC (Zanelli et al., 2017).

★ This research was supported by the EU via H2020-ITN-AWESCO (642 682), by the Federal Ministry for Economic Affairs and Energy (BMWi) via eco4wind (0324125B) and DyConPV (0324166B), and by DFG via Research Unit FOR 2401.

The concept of automatic code generation forms a natural approach to obtain a solver implementation that is compact, self-contained, efficient and possibly customized to the embedded hardware on which it needs to run. However, based on the authors' personal experience (most notably with the `ACADO Code Generation` tool), there can be a high price to pay in flexibility, maintainability and extensibility when developing code generation software for advanced optimal control methods. In addition, such a code generation tool typically does not allow the easy and flexible simulation-based testing of various optimal control problem formulations before deployment. For instance, changing a single algorithmic parameter could result in the need for recompilation of the entire controller. Therefore, in practice, the gap between an initial prototyping phase and the implementation of a code generated solver on the embedded control hardware is often relatively large.

Motivated by the above mentioned limitations of existing software tools, the present paper proposes an alternative framework that enables efficient solvers to be used directly both for prototyping purposes and for real embedded control applications, without the need for a separate code generation phase. This new software package, called `acados`, aims to combine the following objectives:

1. *Flexibility*: interfaces to components, e.g., optimal control problem formulations, remain sufficiently general to treat a wide range of applications
2. *Reproducibility*: the solver implementations form an explicit part of the software package and are not the implicit result of a code generation process

(3) *Modularity*: each of the algorithmic components form a separate module that can therefore be easily replaced by an alternative implementation
(4) *Efficiency*: even though computational efficiency is not our only objective, it remains important for embedded applications with tight timing constraints.

One could expect that targeting general problem formulations in a modular and flexible manner would result in a considerable decrease in computational performance with respect to a code generation approach. We will however show that the proposed software package can remain competitive with code generated solver implementations, while supporting a larger range of algorithms and control applications. One reason for this is the internal use of highly efficient, optimized dense linear algebra routines in the `BLASFEO` software (Frison et al., 2018) for embedded optimization. We remark that `acados` targets small and medium-sized problems; from 10s-1000s of optimization variables in total.

The `acados` package consists of efficient implementations of algorithmic components in `C` and it interfaces to dynamic models through a generic `C` interface. The latter allows one to use a modeling environment with differentiation capabilities that can generate efficient `C` code to evaluate the model equations and forward or adjoint derivatives, e.g., using the `CasADi` (Andersson et al., 2018) framework.

To showcase the software package, we illustrate its computational performance on two nonlinear control examples. The first is solved with a standard SQP-type NMPC scheme using a Gauss-Newton Hessian approximation. In the second simulation example, an efficient implementation of sequential convex quadratic programming (SCQP) is used (Verschueren et al., 2016).

The remainder of the paper is structured as follows. In Section 2, we introduce notation and give a brief introduction to SQP-type NMPC schemes. Then, we introduce SCQP as an algorithmic variation. Subsequently, in Section 3, we give a high-level overview of `acados`. In Section 4, we compare the performance of `acados` and `ACADO Code Generation` using numerical experiments. We conclude the paper in Section 5.

2. SEQUENTIAL QUADRATIC PROGRAMMING FOR NONLINEAR MODEL PREDICTIVE CONTROL

For the purpose of this paper, we are interested in controlling nonlinear dynamic models like described by the ordinary differential equation (ODE)

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$

where $x(t) \in \mathbb{R}^{n_x}$ are the states, $u(t) \in \mathbb{R}^{n_u}$ is the vector of control inputs and time is denoted by t. The initial value for the state is $x(0) = \bar{x}_0$. We assume throughout that the dynamic model $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is continuously differentiable.

A discrete-time approximation of (1) is

$$x_{k+1} = f_{\mathrm{d}}(x_k, u_k), \quad k = 0, 1, \dots, \tag{2}$$

where $f_{\mathrm{d}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is obtained by means of a (one-step) numerical simulation routine.

When using this dynamic model in a discrete-time optimal control setting, constraint (2), together with the initial value constraint $x_0 = \bar{x}_0$, constitute the equality constraints of a discrete-time optimal control problem (OCP). As for the inequality constraints, we make the observation that in real-world control problems, constraints of quadratic form arise naturally. Examples of such constraints are ellipsoidal force constraints, the friction ellipses of tires for road vehicles (Pacejka, 2006) or terminal constraints ensuring asymptotic stability in NMPC schemes (Rawlings et al., 2017). That being said, we present an optimal control problem formulation obtained by a multiple shooting (Bock and Plitt, 1984) discretization given by

$$\begin{aligned}
\underset{\substack{x_0,x_1,\dots,x_N\\u_0,u_1,\dots,u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \frac{1}{2}\|r_k(x_k,u_k)\|_2^2 + \frac{1}{2}\|r_N(x_N)\|_2^2 && \text{(3a)}\\
\text{subject to} \quad & x_0 = \bar{x}_0, && \text{(3b)}\\
& x_{k+1} = f_{\mathrm{d}}(x_k,u_k), \qquad k=0,\dots,N-1, && \text{(3c)}\\
& \underline{c}_k \le c_k(x_k,u_k) \le \bar{c}_k, k=0,\dots,N-1, && \text{(3d)}\\
& \underline{c}_N \le c_N(x_N) \le \bar{c}_N, && \text{(3e)}\\
& \frac{1}{2}\|p_k(x_k,u_k)\|_2^2 \le \rho_k^2, \qquad k=0,\dots,N-1, && \text{(3f)}\\
& \frac{1}{2}\|p_N(x_N)\|_2^2 \le \rho_N^2, && \text{(3g)}
\end{aligned}$$

in which $\underline{c}_k, \bar{c}_k \in \mathbb{R}^{n_c^k}, k = 0, 1, \dots, N$ are the lower and upper bounds, respectively, on the nonlinear inequality constraints $c_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_c^k}, k = 0, 1, \dots, N-1$ and $c_N : \mathbb{R}^{n_x} \to \mathbb{R}^{n_c^N}$. For the quadratic constraints, we have arbitrary nonlinear functions $p_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_p^k}, k = 0, 1, \dots, N-1$ and $p_N : \mathbb{R}^{n_x} \to \mathbb{R}^{n_p^N}$ with corresponding radii $\rho_k, k = 0, 1, \dots, N$. The cost function consists of quadratic penalties on residuals $r_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_r^k}$ and $r_N : \mathbb{R}^{n_x} \to \mathbb{R}^{n_r^N}$. We make the following standard assumption.

Assumption 1. Function f_{d} and functions r_k, c_k, p_k, for $k = 0, 1, \dots, N$, are twice continuously differentiable.

Remark 1. `acados` supports more general OCP formulations than presented in (3). For instance, differential-algebraic equations (DAEs) and discrete-time models, as well as nonlinear terms in the objective are possible but these are outside the scope of this paper.

We introduce Lagrange multipliers for the constraints, with $\lambda_k \in \mathbb{R}^{n_x}, k = 0, \dots, N$ corresponding to the equality constraints, $\mu_{\mathrm{l},k}, \mu_{\mathrm{u},k} \in \mathbb{R}^{n_c^k}, k = 0, \dots, N$ corresponding to the inequality constraints c_k with lower and upper bounds, respectively, and multipliers $\mu_{\mathrm{q},k} \in \mathbb{R}, k = 0, \dots, N$ corresponding to the nonlinear quadratic constraints. By introducing the following shorthands,

$$w = \begin{bmatrix} x_0 \\ u_0 \\ \vdots \\ x_N \end{bmatrix}, \quad \lambda = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_N \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_{\mathrm{u},0} \\ \mu_{\mathrm{l},0} \\ \vdots \\ \mu_{\mathrm{l},N} \end{bmatrix}, \quad \mu_{\mathrm{q}} = \begin{bmatrix} \mu_{\mathrm{q},0} \\ \mu_{\mathrm{q},1} \\ \vdots \\ \mu_{\mathrm{q},N} \end{bmatrix},$$

and

$$c(w) := \begin{bmatrix} c_0(x_0,u_0) - \overline{c}_0 \\ \underline{c}_0 - c_0(x_0,u_0) \\ \vdots \\ c_N(x_N) - \overline{c}_N \\ \underline{c}_N - c_N(x_N) \end{bmatrix},$$

$$F(w) := \begin{bmatrix} x_0 - \overline{x}_0 \\ x_1 - f_\mathrm{d}(x_0,u_0) \\ \vdots \\ x_N - f_\mathrm{d}(x_{N-1},u_{N-1}) \end{bmatrix}$$

$$c_\mathrm{q}(w) := \begin{bmatrix} \frac{1}{2}\|p_0(x_0,u_0)\|_2^2 - \rho_0^2 \\ \vdots \\ \frac{1}{2}\|p_N(x_N)\|_2^2 - \rho_N^2 \end{bmatrix}, r(w) := \begin{bmatrix} r_0(x_0,u_0) \\ r_1(x_1,u_1) \\ \vdots \\ r_N(x_N) \end{bmatrix},$$

the Lagrangian for OCP (3) is defined as

$$\mathcal{L}(w,\lambda,\mu,\mu_\mathrm{q}) := \frac{1}{2}\|r(w)\|_2^2 + \lambda^\top F(w) + \mu^\top c(w) + \mu_\mathrm{q}^\top c_\mathrm{q}(w). \tag{4}$$

With the notation in place, we can now discuss how to solve OCP (3). One possibility would be to use Interior Point (IP) methods. However, Sequential Quadratic Programming (SQP) methods come with a number of benefits (e.g., generalized tangential predictors) over IP methods in an online/embedded context, as discussed in (Diehl et al., 2009) or in (Rawlings et al., 2017). For this reason, we focus on the latter for the remainder of the paper.

2.1 Sequential Quadratic Programming

In order to solve the nonlinear programming problem NLP (3), we use a full-step SQP method

$$w^{[i+1]} = w^{[i]} + \Delta w^{[i]}, \tag{5}$$

where $w^{[i]}$ are the primal variables in the ith SQP iteration. In *embedded* optimization, globalization strategies are less important because a good initial solution guess can always be constructed from one time step to the next. In addition, closed-loop stability results exist for a full-step SQP method in the form of the real-time iteration (RTI) scheme as in (Diehl et al., 2005). Therefore, we won't discuss any globalization strategies like line search or trust regions.

The steps $\Delta w^{[i]}$ are computed by solving a quadratic programming (QP) problem derived from (3). The linearizations of the equality constraints are defined as follows:

$$b_k = f_\mathrm{d}(x_k^{[i]}, u_k^{[i]}) - x_{k+1}^{[i]},$$
$$A_k = \frac{\partial f_\mathrm{d}}{\partial x_k}(x_k^{[i]}, u_k^{[i]}), \quad B_k = \frac{\partial f_\mathrm{d}}{\partial u_k}(x_k^{[i]}, u_k^{[i]}),$$

and inequality constraint matrices

$$C_k = \frac{\partial c_k}{\partial x_k}\left(x_k^{[i]}, u_k^{[i]}\right), \quad D_k = \frac{\partial c_k}{\partial u_k}\left(x_k^{[i]}, u_k^{[i]}\right),$$
$$C_N = \frac{\partial c_N}{\partial x_N}\left(x_N^{[i]}\right),$$
$$P_k = p_k\left(x_k^{[i]}, u_k^{[i]}\right)^\top \frac{\partial p_k}{(\partial x_k, \partial u_k)}\left(x_k^{[i]}, u_k^{[i]}\right),$$
$$P_N = p_N\left(x_N^{[i]}\right)^\top \frac{\partial p_N}{\partial x_N}\left(x_N^{[i]}\right).$$

The resulting QP subproblems are of the form

$$\begin{aligned} \underset{\substack{\Delta x_0,\dots,\Delta x_N \\ \Delta u_0,\dots,\Delta u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix}\Delta x_k \\ \Delta u_k\end{bmatrix}^\top H_k \begin{bmatrix}\Delta x_k \\ \Delta u_k\end{bmatrix} \\ & + \sum_{k=0}^{N-1} r_k(x_k^{[i]}, u_k^{[i]})^\top \frac{\partial r_k(x_k^{[i]}, u_k^{[i]})}{(\partial x_k, \partial u_k)} \begin{bmatrix}\Delta x_k \\ \Delta u_k\end{bmatrix} \\ & + \frac{1}{2}\Delta x_N^\top H_N \Delta x_N + r_N(x_N^{[i]})^\top \frac{\partial r_N(x_N^{[i]})}{\partial x_N}\Delta x_N \\ \text{subject to} \quad & \\ & \Delta x_0 = \overline{x}_0 - x_0^{[i]}, \\ & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k, \qquad k = 0,\dots,N\text{-}1, \\ & \underline{c}_k - c_k(x_k^{[i]}, u_k^{[i]}) \le C_k \Delta x_k + D_k \Delta u_k, \qquad k = 0,\dots,N\text{-}1, \\ & C_k \Delta x_k + D_k \Delta u_k \le \overline{c}_k - c_k(x_k^{[i]}, u_k^{[i]}), \qquad k = 0,\dots,N\text{-}1, \\ & \underline{c}_N - c_N(x_N^{[i]}) \le C_N \Delta x_N \le \overline{c}_N - c_N(x_N^{[i]}), \\ & P_k \begin{bmatrix}\Delta x_k \\ \Delta u_k\end{bmatrix} \le \rho_k^2 - \frac{1}{2}\|p_k(x_k^{[i]}, u_k^{[i]})\|_2^2, \qquad k = 0,\dots,N\text{-}1, \\ & P_N \Delta x_N \le \rho_N^2 - \frac{1}{2}\|p_N(x_N^{[i]})\|_2^2, \end{aligned} \tag{6}$$

with Hessian matrices $H_k \in \mathbb{R}^{(n_x+n_u)\times(n_x+n_u)}$ and $H_N \in \mathbb{R}^{n_x \times n_x}$.

QP problem (6) can now be solved by a structure-exploiting QP solver, like `FORCES Pro` (Domahidi and Perez, 2013) or `HPMPC` (Frison et al., 2014), possibly after a *partial condensing* step (Axehill, 2015). Alternatively, the QP problem (6) can be reformulated using a *condensing* step (Bock and Plitt, 1984) to eliminate the state variables and solved by a dense QP solver like `qpOASES` (Ferreau et al., 2014). In addition to the primal steps $\Delta w^{[i]}$, estimates of the dual variables $\lambda^{[i]}, \mu^{[i]}$ and $\mu_\mathrm{q}^{[i]}$ are available from the QP solution.

We note that, in the case of a nonlinear least-squares objective as in (3), a Gauss-Newton Hessian approximation

$$H_k^\mathrm{GN} = \left(\frac{\partial r_k}{(\partial x_k, \partial u_k)}\right)^\top \left(\frac{\partial r_k}{(\partial x_k, \partial u_k)}\right), \tag{7}$$

$$H_N^\mathrm{GN} = \left(\frac{\partial r_N}{\partial x_N}\right)^\top \left(\frac{\partial r_N}{\partial x_N}\right) \tag{8}$$

is commonly used in practice, as the Hessian approximation matrices are always positive semi-definite, and they are cheap to compute, compared to the exact Hessian. Indeed, the Jacobians in (7)-(8) are readily available from the computations of the cost gradient. Furthermore, for a linear least-squares cost function, i.e.,

$$r_k(x_k, u_k) = y_k - V_k \begin{bmatrix} x_k \\ u_k \end{bmatrix},$$
$$r_N(x_N) = y_N - V_N x_N,$$

with $V_k \in \mathbb{R}^{n_r^k \times (n_x+n_u)}$, $y_k \in \mathbb{R}^{n_r^k}$ and $V_N \in \mathbb{R}^{n_r^N \times n_x}$, $y_N \in \mathbb{R}^{n_r^N}$, the Gauss-Newton Hessian approximations are constant matrices that can be computed as

$$H_k^\mathrm{LLS} = V_k^\top V_k, \quad k = 0, 1, \dots, N.$$

It is well known that a Gauss-Newton Hessian approximation, although it only induces a locally linear convergence rate, performs very well when the residuals $r_k(x_k^{[i]}, u_k^{[i]}), r_N(x_N^{[i]})$ are small (Bock et al., 2000).

Quasi-Newton methods make use of different Hessian approximation strategies; the Hessian is updated with matrices derived from first-order derivative information. Examples include symmetric rank 1 (SR1) and BFGS updates. However, as mentioned in (Quirynen, 2017), these kind of Hessian updates are less useful in embedded optimization, as the rate of convergence potentially fluctuates.

2.2 Sequential Convex Quadratic Programming (SCQP)

In the following, we explain how we exploit the partial convexity that is introduced by the quadratic inequality constraints in NLP (3). In (Verschueren et al., 2016), a slightly more general treatment can be found.

We start by looking at the contributions that make up the exact Hessian of the Lagrangian, as defined in (4):

$$\begin{aligned}
&\nabla_w^2 \mathcal{L}(w, \lambda, \mu, \mu_\mathrm{q}) = \\
&\sum_k \left(\frac{\partial r_k}{\partial w}(w)\right)^\top \left(\frac{\partial r_k}{\partial w}(w)\right) && \text{(9a)}\\
&+ \sum_k \mu_{\mathrm{q},k} \left(\frac{\partial p_k}{\partial w}(w)\right)^\top \left(\frac{\partial p_k}{\partial w}(w)\right) && \text{(9b)}\\
&+ \sum_k \sum_i r_{k,i}(w)\, \nabla_w^2 r_{k,i}(w) && \text{(9c)}\\
&+ \sum_k \sum_i \mu_{\mathrm{q},k} p_{k,i}(w)\, \nabla_w^2 p_{k,i}(w) && \text{(9d)}\\
&+ \sum_k \sum_i \mu_{\mathrm{u},k,i} \nabla_w^2 c_{k,i}(w) - \mu_{\mathrm{l},k,i} \nabla_w^2 c_{k,i}(w) && \text{(9e)}\\
&- \sum_k \sum_i \lambda_{k+1,i} \nabla_w^2 f_{\mathrm{d},i}(w). && \text{(9f)}
\end{aligned}$$

We note that the Gauss-Newton Hessian approximation only takes into account the terms (9a).

Comparing the quadratic form of the cost function in (3) and the quadratic constraints in (3f)-(3g), we can devise another Hessian approximation that also takes into account the terms (9b) from the quadratic inequality constraints, which reads as

$$H_k^{\mathrm{SCQP}} = H_k^{\mathrm{GN}} + \mu_{\mathrm{q},k}^{[i]} \left(\frac{\partial p_k}{(\partial x_k, \partial u_k)}\right)^\top \left(\frac{\partial p_k}{(\partial x_k, \partial u_k)}\right), \tag{10}$$

and

$$H_N^{\mathrm{SCQP}} = H_N^{\mathrm{GN}} + \mu_{\mathrm{q},N}^{[i]} \left(\frac{\partial p_N}{\partial x_N}\right)^\top \left(\frac{\partial p_N}{\partial x_N}\right), \tag{11}$$

where the Jacobians are evaluated at iterate i: $(x_k^{[i]}, u_k^{[i]})$.

Note that, in contrast to the Gauss-Newton Hessian approximation, we need the current guess of the optimal Lagrange multipliers, $\mu_{\mathrm{q},k}^{[i]}$. However, the benefits of using the SCQP Hessian are similar to that of the Gauss-Newton Hessian. It is cheap to evaluate Eq. (10) as the Jacobians $\partial p_k/(\partial x_k, \partial u_k)$ are also required in the computation of the matrix P_k in (6).

There is one particular advantage of SCQP over the standard Gauss-Newton Hessian approximation, namely that it is not only positive semi-definite but in fact the following property holds:

Property 1. $H_k^{\mathrm{SCQP}} \succeq H_k^{\mathrm{GN}}$, for $k = 0, 1, \ldots, N$.

This follows from the Karush-Kuhn-Tucker (KKT) conditions ($\mu_{\mathrm{q},k}^{[i]} \geq 0$) on the QP subproblem and the positive semi-definiteness of the last terms in (10) and (11). The need for the multipliers $\mu_{\mathrm{q},k}^{[i]}$ does not introduce extra computations in the SCQP algorithm, as these values are readily available from the QP solution.

Property 1 is beneficial in the sense that the SCQP Hessian approximation is often, though not always, closer to the exact Hessian of the Lagrangian than a Gauss-Newton approximation: it takes into account terms (9a) and (9b). In problems with quadratic constraints, this can have a positive effect on local convergence. A numerical example of this will be given in Section 4, after we present the software.

Remark 2. A possible alternative to SCQP in the setting of this section is sequential convex programming (SCP), cf. (Tran-Dinh and Diehl, 2010), where in each iteration a quadratically-constrained QP (QCQP) is solved. Both methods, SCQP and SCP, can be shown to have the same asymptotic convergence rate. The advantage of SCQP is that it can build on any state-of-the-art embedded QP solver, which allows it to benefit from the many competitive codes in this class, and also helps a modular implementation of both SQP-type and SCQP-type methods in one environment.

3. A MODULAR SOFTWARE PACKAGE FOR EMBEDDED OPTIMIZATION-BASED CONTROL

Good software for advanced optimization-based control algorithms is crucial for using NMPC in practical applications. One example of a software package that was successfully employed in real experiments is the `ACADO Code Generation` tool (Houska et al., 2011). Using code generation, fast embedded solvers are obtained from a high-level description of the control problem. Examples of real-world experiments with `ACADO Code Generation` can be found in (Debrouwere et al., 2014), (Besselmann et al., 2015), and (Abdollahpouri et al., 2017).

However, there are several drawbacks to the `ACADO Code Generation`. In general, from a software point of view, it is more difficult to extend a code generator to generate new algorithms. Secondly, since the generated code is written for compilers, not for humans, it is more cumbersome to read, understand and debug. Moreover, code generation often results in larger binaries, as the code becomes longer by introducing code optimizations like loop unrolling.

The advent of a new high-performance dense linear algebra package for embedded computations, `BLASFEO` (Frison et al., 2018), makes it possible to circumvent or even do away with code generation altogether. The highly efficient linear algebra routines, optimized for a range of computer architectures, often outperform code generated linear algebra, as it explicitly targets architecture-specific features (as e.g., register space, execution pipelines, vector instructions, cache policy).

Table 1. Overview of the software modules present in acados.

Module	Variants
OCP QP	qpOASES qpDUNES HPMPC HPIPM OOQP
Condensing	Full condensing Partial (block) condensing
Integrator	Explicit Euler Runge-Kutta 4 Implicit Runge-Kutta (Gauss-Legendre)
OCP NLP	Gauss-Newton SQP SCQP
Nonlinear function	CasADi code-generated functions Hand-written C-code

With the new software package acados, which builds on top of the BLASFEO linear algebra framework, we offer data types and efficient algorithms for embedded optimization. acados is an open source software (license LGPLv3) implemented in C and available online from (Acados, 2018). It is conceived as a modular software package, where different implementations of a module could be easily switched. As an example, a simulation routine implementing the classical Runge-Kutta method (RK4) can be easily replaced by an implicit Runge-Kutta method. In Table 1, we provide an overview of the different modules currently present in acados.

Interfaces to third-party solvers are also provided, mainly for solving QPs: qpOASES (Ferreau et al., 2014), qpDUNES (Frasch et al., 2015), HPMPC (Frison et al., 2014), HPIPM (Frison, 2017) and OOQP (Gertz and Wright, 2003). HPIPM also provides an efficient implementation of condensing and partial condensing algorithms (Frison et al., 2016), which are systematically used in acados as pre-processing steps in the solution of the QPs. For the nonlinear functions in constraints and objective, we interface with functions code generated by CasADi (Andersson et al., 2018), while leaving the option to use hand-written models. We package these external codes behind a uniform interface. As such, changing between two 'variants' of a module is completely transparent. We adopt the same interface for the in-house solvers also mentioned in Table 1.

There are two main advantages of acados over ACADO Code Generation from a user point of view: first, the modular design of acados is such that users can pick only those modules that they actually need. For example, if one wants to implement linear MPC, it is possible to only make use of the OCP QP module. To some extent, this was possible for simulation routines in ACADO Code Generation, but a structured way of independently accessing functionalities implemented by different modules was not present in the framework. Furthermore, a small code change (e.g. the length of the horizon N) does not require a full regeneration and recompilation of the solver. In this case, acados needs not to be recompiled at all, since any parameter can be changed easily.

From the developer's perspective, we identify two major improvements. First and foremost, modules are responsible for their memory management. This eliminates the need for data structures that are globally accessible, and draws clear interfaces between modules. For instance, the integrator shall not share memory with the QP solver, nor should it access the internal representation in, e.g., the condensing routines. This makes it more straightforward for developers to reason about their code. Additionally, the interfaces to the modules are meant to be *uniform*, as mentioned above. This facilitates extension of the software, e.g., by creating more implementations of modules based on other third-party solvers in the future, by using the same *uniform* interfaces.

To conclude this section, we emphasize that acados is not a reimplementation of ACADO Toolkit. Instead, it is a fully novel package, based on the general principles of ACADO Code Generation, but not on the syntax nor the design. For these reasons, the two packages are incompatible.

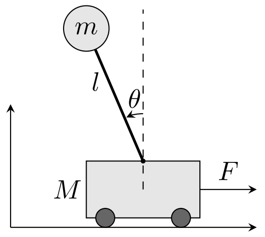

Fig. 1. Inverted pendulum on top of a moving cart.

4. NUMERICAL RESULTS

In this section, we illustrate the use of acados for the classical nonlinear control example of the swing-up of an inverted pendulum on a cart.

4.1 Inverted Pendulum Model

In Figure 1, a sketch of an inverted pendulum with mass m [kg] and length l [m] on a cart with mass M [kg] is given. We can control the movement of the pendulum with the horizontal force F [N] on the cart. We model this system with state vector $x(t) = [p(t), v(t), \theta(t), \omega(t)]^\top$, with horizontal position p [m] and velocity v [m/s], and angle θ [rad] and angular velocity ω [rad/s], respectively. The control is $u(t) = F(t)$. The continuous-time model equations then read as

$$\dot{p} = v \tag{12a}$$

$$\dot{v} = \frac{-ml\sin(\theta)\omega^2 + mg\cos(\theta)\sin(\theta) + F}{M + m - m(\cos(\theta))^2} \tag{12b}$$

$$\dot{\theta} = \omega \tag{12c}$$

$$\dot{\omega} = \frac{-ml\cos(\theta)\sin(\theta)\omega^2 + F\cos(\theta) + (M+m)g\sin(\theta)}{l(M + m - m(\cos(\theta))^2)} \tag{12d}$$

where we chose the parameter values $M = 1\,\text{kg}$, $m = 0.1\,\text{kg}$, $g = 9.81\,\text{m/s}^2$, and $l = 0.8\,\text{m}$.

4.2 NMPC Results using Gauss-Newton SQP

In an online setting, we repeatedly measure or estimate the state of the system and subsequently feed back an optimal control which is taken from the solution of the following OCP:

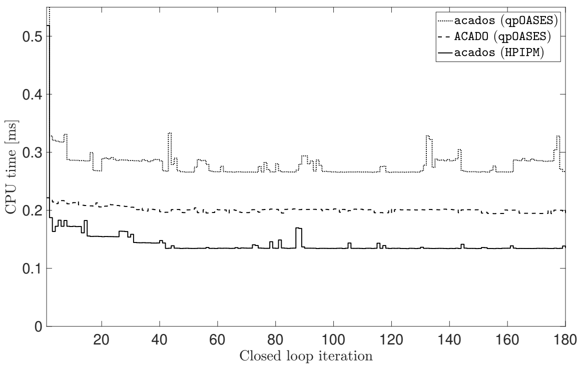

Fig. 2. Computation times for closed-loop NMPC, implemented with `acados` versus `ACADO Code Generation`.

$$\begin{aligned} \underset{\substack{x_0,x_1,\ldots,x_N\\ u_0,u_1,\ldots,u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} u_k^2 \\ \text{subject to} \quad & x_0 = \overline{x}_0, \\ & x_{k+1} = f_{\text{pend}}(x_k,u_k), \quad k=0,\ldots,N-1, \\ & x_N = 0, \end{aligned} \tag{13}$$

where $f_{\text{pend}} : \mathbb{R}^4 \times \mathbb{R} \to \mathbb{R}^4$ is a discrete-time representation of ODE (12) obtained by using an RK4 integrator with discretization interval of 0.05 s and $N = 40$ intervals.

Running the NMPC in closed loop, we start with the pendulum hanging down ($\overline{x}_0 = [0, 0, \pi, 0]^\top$). We employ the real-time iteration (RTI) scheme (Diehl et al., 2002): in each sampling time, we approximately solve OCP (13) by performing only one SQP iteration (cf. (6)). We do not perform shifting of the primal/dual variables between sampling times. We initialize the solver in the starting point, i.e., $x_k^{[0]} = \overline{x}_0, k = 0,\ldots,N$ and $u_k^{[0]} = 0, k = 0,\ldots,N-1$.

In the following, we look at the performance of two NMPC codes; one is generated by the `ACADO Code Generation` tool, one is implemented with `acados`. They both yield exactly the same solutions in closed loop, as the same algorithms are employed. The default parameters are chosen in both tools, unless otherwise stated.

We compare the computation times of the two NMPC codes in Figure 2. All measurements are carried out on a MacBook Pro with a 2.5 GHz Intel Core i7-4870HQ processor. We see that the median computation time (266 μs) of `acados`, for the same underlying QP solver `qpOASES` v3.1 is approximately 20% longer than that of `ACADO` (229 μs). For a different QP solver, namely `HPIPM`, in combination with partial condensing with block size 4, the solve time of `acados` dips below that of `ACADO`, by 40% (median computation time is 137 μs). We remark that more extensive benchmarks need to be carried out for a proper comparison between the two tools, which is outside the scope of the present paper.

4.3 NMPC Results using SCQP

In this section, we present results on using SCQP, as explained in Section 2.2, within `acados`. No comparison with `ACADO` is possible here, because SCQP did not exist as a feature in `ACADO` and would be cumbersome to implement.

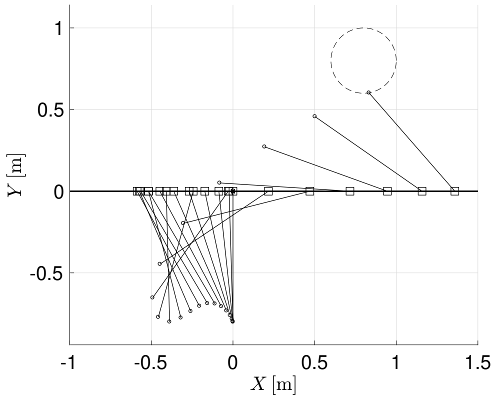

Fig. 3. SCQP on the pendulum example (cf. OCP (14)).

To illustrate this algorithmic variant, we slightly modify the OCP to

$$\begin{aligned} \underset{\substack{x_0,x_1,\ldots,x_N\\ u_0,u_1,\ldots,u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} u_k^2 \\ \text{subject to} \quad & x_0 = \overline{x}_0, \\ & x_{k+1} = f_{\text{pend}}(x_k,u_k), \quad k=0,\ldots,N-1, \\ & \left\| \begin{bmatrix} p_N - l\sin(\theta_N) - l \\ l\cos(\theta_N) - l \end{bmatrix} \right\|_2^2 \le \rho^2, \end{aligned} \tag{14}$$

such that the position of the pendulum at the end of the trajectory is inside a circle of radius ρ [m] around the point (l, l). With this quadratic constraint, we associate the Lagrange multiplier μ_{q}.

For different values of ρ, we have different Hessian approximations for SCQP, as the optimal multiplier μ_{q} varies. The Gauss-Newton approximation is constant for all ρ.

For some ρ, the Gauss-Newton Hessian approximation is not sufficiently good for a full-step SQP method to converge, not even when initialized close to a local minimizer. In contrast, SCQP converges for all ρ, depending where the solver is initialized. As an example, look at $\rho = 0.04$ m. We initialize the full-step S(C)QP method at $x_k^{[0]} = 0, k = 0,\ldots,N$ and $u_k^{[0]} = 0, k = 0,\ldots,N-1$. SCQP converges in 38 iterations, with a total computation time of 5.5 ms. The optimal control trajectory is shown in Figure 3. SQP with a Gauss-Newton Hessian approximation, however, does not converge, regardless where it is initialized (except when initializing exactly at the solution).

5. CONCLUSION

In this paper, we introduced a new software package for embedded optimization, called `acados`. We showed some preliminary numerical experiments, and an implementation of a new algorithmic idea, namely SCQP.

Subject of future work is a more extensive comparison with other solvers, including benchmarks, extension with a nonlinear interior point method, and interfaces to higher-level programming languages like `Python` and `MATLAB` in order to capture a larger user base.

REFERENCES

Abdollahpouri, M., Takács, G., and Rohal'-Ilkiv, B. (2017). Real-time moving horizon estimation for a vibrating active cantilever. *Mechanical Systems and Signal Processing*, 86, 1–15.

Acados (2018). A modular software package for embedded optimization. `www.acados.org`.

Andersson, J.A.E., Gillis, J., Horn, G., Rawlings, J.B., and Diehl, M. (2018). CasADi–a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*. (accepted).

Axehill, D. (2015). Controlling the level of sparsity in MPC. *Systems & Control Letters*, 76, 1–7.

Besselmann, T.J., de moortel, S.V., Almér, S., Jörg, P., and Ferreau, H.J. (2015). Model predictive control in the multi-megawatt range. *IEEE Transactions on Industrial Electronics*, 63(7), 4641–4648.

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

Bock, H.G. and Plitt, K.J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the IFAC World Congress*, 242–247. Pergamon Press.

Debrouwere, F., Vukov, M., Quirynen, R., Diehl, M., and Swevers, J. (2014). Experimental validation of combined nonlinear optimal control and estimation of an overhead crane. In *Proceedings of the IFAC World Congress*, 9617–9622.

Diehl, M., Bock, H.G., Schlöder, J.P., Findeisen, R., Nagy, Z., and Allgöwer, F. (2002). Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4), 577–585.

Diehl, M., Ferreau, H.J., and Haverbeke, N. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. In L. Magni, M. Raimondo, and F. Allgöwer (eds.), *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, 391–417. Springer.

Diehl, M., Findeisen, R., Allgöwer, F., Bock, H.G., and Schlöder, J.P. (2005). Nominal stability of the real-time iteration scheme for nonlinear model predictive control. *IEE Proc.-Control Theory Appl.*, 152(3), 296–308.

Domahidi, A. and Perez, J. (2013). FORCES professional. http://embotech.com/FORCES-Pro.

Ferreau, H.J., Almer, S., Verschueren, R., Diehl, M., Frick, D., Domahidi, A., Jerez, J.L., Stathopoulos, G., and Jones, C. (2017). Embedded optimization methods for industrial automatic control. In *Proceedings of the IFAC World Congress*.

Ferreau, H.J., Kirches, C., Potschka, A., Bock, H.G., and Diehl, M. (2014). qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4), 327–363.

Frasch, J.V., Sager, S., and Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computations*, 7(3), 289–329.

Frison, G. (2017). HPIPM: High performance interior point methods. https://github.com/giaf/hpipm.

Frison, G., Kouzoupis, D., Jørgensen, J.B., and Diehl, M. (2016). An efficient implementation of partial condensing for nonlinear model predictive control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 4457–4462.

Frison, G., Kouzoupis, D., Sartor, T., Zanelli, A., and Diehl, M. (2018). BLASFEO: Basic linear algebra subroutines for embedded optimization. *ACM Transactions on Mathematical Software (TOMS)*. (accepted).

Frison, G., Sorensen, H.B., Dammann, B., and Jørgensen, J.B. (2014). High-performance small-scale solvers for linear model predictive control. In *Proceedings of the European Control Conference (ECC)*, 128–133.

Gertz, E.M. and Wright, S.J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1), 58–81.

Houska, B., Ferreau, H.J., and Diehl, M. (2011). An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 47(10), 2279–2285.

Kalmari, J., Backman, J., and Visala, A. (2015). A toolkit for nonlinear model predictive control using gradient projection and code generation. *Control Engineering Practice*, 39, 56–66.

Mattingley, J. and Boyd, S. (2009). *Convex Optimization in Signal Processing and Communications*, chapter Automatic Code Generation for Real-Time Convex Optimization, 1–41. Cambridge University Press.

Ohtsuka, T. and Kodama, A. (2002). Automatic code generation system for nonlinear receding horizon control. *Transactions of the Society of Instrument and Control Engineers*, 38(7), 617–623.

Pacejka, H.B. (2006). *Tyre and Vehicle Dynamics*. Elsevier.

Quirynen, R. (2017). *Numerical Simulation Methods for Embedded Optimization*. Ph.D. thesis, KU Leuven and University of Freiburg.

Rawlings, J.B., Mayne, D.Q., and Diehl, M.M. (2017). *Model Predictive Control: Theory, Computation, and Design*. Nob Hill, 2nd edition edition.

Tran-Dinh, Q. and Diehl, M. (2010). Local convergence of sequential convex programming for nonconvex optimization. In M.Diehl, F.Glineur, E. Jarlebring, and W. Michiels (eds.), *Recent advances in optimization and its application in engineering*, 93–103. Springer-Verlag.

Verschueren, R., van Duijkeren, N., Quirynen, R., and Diehl, M. (2016). Exploiting convexity in direct optimal control: a sequential convex quadratic programming method. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Zanelli, A., Domahidi, A., Jerez, J.L., and Morari, M. (2017). FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*.

Zometa, P., Kögel, M., and Findeisen, R. (2013). muAO-MPC: A free code generation tool for embedded real-time linear model predictive control. In *2013 American Control Conference*, 5320–5325.