Efficient NMPC for nonlinear models with linear subsystems

Rien Quirynen, Sébastien Gros and Moritz Diehl

Abstract—Real-time optimal control algorithms for fast, mechatronic systems need to be run on embedded hardware and they need to respect tight timing constraints. When using nonlinear models, the simulation and generation of sensitivities forms a computationally demanding part of any algorithm. Automatic code generation of Implicit Runge-Kutta (IRK) methods has been shown to reduce its CPU time significantly. However, a typical model also shows a lot of structure that can be exploited in a rather elegant and efficient way. The focus of this paper is on nonlinear models with linear subsystems. With the proposed model formulation, the new auto generated integrators can be considered a powerful generalization of other solvers, e.g. those that support quadrature variables. A speedup of up to 5−10 is shown in the integration time for two examples from the literature.

Keywords: structure exploitation, code generation, IRK methods, NMPC, embedded optimization

I. INTRODUCTION

Nonlinear Model Predictive Control (NMPC) [1] and Moving Horizon Estimation (MHE) [2] are popular approaches for real-time optimal control and estimation, since they can explicitly handle constraints and nonlinear dynamics. In the case of a system with fast dynamics, the high computational burden forms a major challenge. An Optimal Control Problem (OCP) needs to be solved at each sampling time, respecting the hard timing constraints which are imposed by real-time applications. Recent algorithmic progress [3], [4] allows to consider NMPC and MHE also for fast systems. Among the available online algorithms, the Real-Time Iteration (RTI) scheme [5] has been proposed as a highly competitive approach. It is an SQP-type algorithm that uses a Gauss-Newton Hessian approximation.

In addition to using an online algorithm, efficient implementations are needed to allow running it in real-time on embedded control hardware. One way to achieve this is by automatic code generation, i.e. by exporting a customized solver. Significant improvements in the computation time can then be obtained by removing unnecessary computations, optimizing the memory access and cache usage, and by exploiting the problem structure. This idea has become quite popular for e.g. convex optimization [6]. It would be possible to assemble a complete algorithm for solving an OCP, starting from components of which each could be code generated when advantageous. This is the approach pursued by the ACADO code generation tool, which exports highly efficient C-code based on the RTI scheme [7].

A rather important component in case of NMPC is the one that handles integration and sensitivity generation for the nonlinear model. It forms typically a major or even dominating factor in the total computation time, while the accuracy of the derivatives also strongly defines the success of the optimizer. Because of the need for a more or less deterministic runtime in case of real-time applications, adaptivity is avoided where possible. The assumption in this paper is therefore that the step size and order of the integration method are fixed. It has been shown that code generation for Implicit Runge-Kutta (IRK) methods with a tailored approach for computing the sensitivities can be very efficient [8]. This is important because of the attractive properties of A-stable IRK methods and their natural extension from systems of Ordinary Differential Equations (ODE) to Differential Algebraic Equations (DAE) of index 1.

In this paper, the following OCP formulation is addressed

$$\min_{x(t),u(\tau)} \int_t^{t+T} \left( \|x(\tau) - x_{ref}(\tau)\|^2 + \|u(\tau) - u_{ref}(\tau)\|^2 \right) d\tau$$

subject to

$$x(t) = \bar{x}_i,$$
$$\dot{x}(\tau) = f(x(\tau),u(\tau)),$$
$$y \leq u(\tau) \leq \bar{u},$$
$$\bar{x} \leq x(\tau) \leq \bar{x}, \quad \forall \tau \in [t,t+T],$$

in which $x$ and $u$ are the states and controls, $x_{ref}$ and $u_{ref}$ are the state and control references, $t$ denotes the current time and $T$ the length of the prediction horizon. Of particular interest is the nonlinear function $f$ which defines the system dynamics. It is known that the models used in applications often have a specific structure resulting in e.g. a sparse Jacobian matrix. A scalable way to handle this is by using sparse solutions for the linear algebra operators. It will however create little to no gain for small to medium scale systems which are targeted by this paper. The goal is therefore to propose an alternative approach for structure exploitation, that will be shown to perform very well. The
addressed model class contains as a subclass the Wiener-Hammerstein models which are since a long time used in system identification [10].

Contribution: This paper presents algorithms to exploit a frequently occurring model structure, resulting in very efficient auto generated integrators.

The paper is organized as follows. In Section II, a three stage model formulation will be proposed. Section III summarizes the used framework of collocation methods with sensitivity generation. Then, the issue of how to exploit the three stage structure in these IRK methods, is addressed by Section IV. Finally, some real-world applications are used in Section V to illustrate the structure exploiting integrators and their performance.

II. THE THREE STAGE STRUCTURE

In this section, a typical structure consisting of three different stages in the system of differential equations is discussed. Note that any separation in subsystems allows to reduce the computational complexity of the integrators, but it becomes tractable once there can be defined such a general structure that then covers many of the interesting cases.

A. Model formulation

In the case of an explicit ODE system such as in (1), one would often recognize one or more of the following three subsystems in this specific order

\[ \dot{x}^{[1]} = A_1x^{[1]} + B_1u, \quad \text{(linear input system)} \]

\[ \dot{x}^{[2]} = f_2(x^{[1]}, x^{[2]}, u), \quad \text{(nonlinear system)} \]

\[ \dot{x}^{[3]} = A_3x^{[3]} + f_3(x^{[1]}, x^{[2]}, u), \quad \text{(linear output system)} \]

with the matrices \( A_1, B_1 \) and \( A_3 \) and the nonlinear functions \( f_2 \) and \( f_3 \) defining the subsystems. Figure 1 illustrates this chain of three subsystems. In what follows, \( N_1, N_2 \) and \( N_3 \) denote the number of differential states in each of the three subsystems. Let us generalize this structure to an implicit DAE system of index 1 as follows:

\[
\begin{align*}
C_1 \dot{x}^{[1]} &= A_1x^{[1]} + B_1u, \\
0 &= f_2(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u), \\
C_3 \dot{x}^{[3]} &= A_3x^{[3]} + f_3(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u),
\end{align*}
\]

with invertible matrices \( C_1 \) and \( C_3 \) and function \( f_2 \) satisfying that the Jacobian \( \frac{\partial f_2}{\partial (x, \dot{x})} \) is invertible.

B. Motivation

The introduced structure is not something that comes up only in rare occasions. These different stages often arise naturally when modeling for control. A linear input system such as in (2a) can for example result from

- some partially linear dynamics which are independent of the states in the nonlinear equations. A classical example would be the double integrator

\[
\begin{align*}
\dot{x}_1 &= u \\
\dot{x}_2 &= x_1
\end{align*}
\]

but any linear variant of this is possible.

- implementing a filter on the controls or input states. In optimal control, it can e.g. be useful to penalize any high frequency content. Therefore, a High-Pass RC filter can be used such as

\[
\frac{dx(t)}{dt} = -\omega_c x(t) + \frac{du(t)}{dt}
\]

where \( \omega_c = 2\pi f_C \) in which \( f_C \) is the cutoff frequency.

The linear output system in (2c) is somewhat similar to the input system, but with a nonlinear relation with respect to the previous states and control inputs. An output system can therefore result from some partially linear dynamics which also depend on the previous states or it can implement a filter for these states. The matrix \( A_3 \) will generally be nonzero. But in the specific case that \( A_3 = 0 \) and \( C_3 \) is an identity matrix, Equation (2c) reduces to

\[
x^{[3]} = f_3(x^{[1]}, x^{[2]}, x^{[1]}, x^{[2]}, z, u)
\]

which have been presented before as quadrature variables [11]. They are typically used to formulate objective and constraint functions in an OCP.

III. AUTO GENERATED IRK METHODS

This section compactly describes the general implementation of auto generated IRK methods, presented earlier in [8] and [9]. Their formulation is handled in Subsection III-A, for ODE as well as DAE systems of index 1. Some comments on the implementation are repeated in Subsection III-B and the efficient generation of continuous output and sensitivities are respectively described by Subsection III-C and III-D.

A. Formulation

The general IRK methods from [9] solve the following Initial Value Problem (IVP) over a certain time:

\[
\begin{align*}
0 &= f(t, x(t), x(t), z(t), u(t)), \\
x(0) &= x_0,
\end{align*}
\]

with \( x(t) \) a vector of \( N_x \) differential states, \( \dot{x}(t) \) the corresponding time derivatives, \( z(t) \) a vector of \( N_z \) algebraic states and \( u(t) \) a vector of control inputs so that \( f \) defines a system of \( N_x + N_z \) equations. Note that \( N_x = N_x^{[1]} + N_x^{[2]} + N_x^{[3]} \) in case of the structure in (2). These methods can handle models ranging from explicit ODE to fully implicit DAE systems. The Jacobian matrix \( \frac{\partial f}{\partial (x, \dot{x})} \) needs to be invertible, so that the
algebraic states $z$ and the differential state derivatives $\dot{x}$ are uniquely defined, i.e. the DAE system should be of index 1. Because of their high order of accuracy and good stability properties, the focus of this paper will be on A-stable IRK methods such as the Gauss methods [12]. Applied to the system in (6) for the integration from $t_n$ until $t_{n+1} = t_n + h$, an $s$-stage IRK method can be formulated as follows:

$$0 = f \left( t_n + c_i h, k_i, x_n + h \sum_{j=1}^{s} a_{ij} k_j, Z_i \right), \quad i = 1, \ldots, s, \quad (7)$$

$$x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i,$$

with $Z_i$ the stage values of the algebraic states, $k_i$ the stage values of the differential state derivatives and the coefficients $c_i$, $b_i$, and $a_{ij}$ are defined by a Butcher table. The method results in a nonlinear system of $s(N_z + N_c)$ equations that can be solved using a Newton method. Using this scheme, consistency of the state values is only assured at the $s$ collocation nodes. For stiffly accurate methods such as the Radau IIA methods, the endpoint is included in these nodes.

**B. Implementation**

With real-time applications in mind, the ACADO code generation tool exports a customized integrator with a deterministic runtime. The step size, the order of the method and the number of Newton iterations are therefore fixed. Let us write the nonlinear system from (7) as $G(x_n, K) = 0$, resulting in the following Newton-type iterations:

$$M = \frac{\partial G}{\partial K}(x_n, K^{[0]}),$$

$$K^{[i]} = K^{[i-1]} - M^{-1} G(x_n, K^{[i-1]}), \quad i = 1, \ldots, L, \quad (8)$$

where $M$ is the evaluation of the Jacobian $\frac{\partial G}{\partial K}$ in the initialization point and the variables $K = (k_1, \ldots, k_s, Z_1, \ldots, Z_s)$. Note that one Jacobian evaluation and a small amount of iterations $L$ is sufficient if this initial $K^{[0]}$ is good enough. The evaluation of the Jacobian can be done using Algorithmic Differentiation (AD) [13]. A custom linear solver can e.g. be based on the LU decomposition of the matrix $M$.

**C. Continuous output**

Promising possibilities of auto generated IRK methods with continuous output have been presented in [9]. In the case of collocation methods, a specific family of IRK methods, this continuous extension comes naturally. Using the collocation variables in $K$, a separate polynomial is defined for the states $x(t)$, $\dot{x}(t)$ and $z(t)$. In the case of an $s$-stage IRK method of order $p$, the order of this approximation is $p^s = \min(p, s+1)$ for $x(t)$ and $p^s - 1$ for $\dot{x}(t)$ and $z(t)$. Output functions as general as

$$y = \psi(t, \dot{x}(t), x(t), z(t)), \quad (9)$$

can efficiently be evaluated on an arbitrarily fine grid.

**D. Sensitivity generation**

In the context of dynamic optimization, sensitivities with respect to all the independent variables $w_n = (x, u)_n$ are needed in addition to the simulated values of states and outputs. In the case of extra outputs, forward techniques to propagate the sensitivities are recommended. A thorough discussion on such techniques of sensitivity generation for IRK methods can be found in [14]. Concluding from that discussion, the most efficient way is to apply the Implicit Function Theorem (IFT) to $G(w_n, K) = 0$, resulting in

$$\frac{dK}{dw_n} = -\frac{\partial G^{-1}}{\partial K} \frac{\partial G}{\partial w_n}, \quad (10)$$

which can then be used to compute the sensitivities of the states and outputs. This direct approach provides very accurate sensitivities, which is important for optimization. The Jacobian evaluation in (10) and its factorization can be reused in (8) for the next integration step [8]. Algorithm 1 compactly describes the resulting implementation.

**Algorithm 1 The implementation of one step**

**Input:** $w_n$, initial $K^{[0]}$, LU factorization of $M$

**Output:** $(x, \dot{x}/\partial w_n)_{t_{n+1}}$ and $(z, \partial z/\partial w_n)_{t_n}$

1. for $i = 1 \rightarrow L$
   2. $K^{[i]} \leftarrow K^{[i-1]} - M^{-1} G(w_n, K^{[i-1]})$
   3. end for

4. $x_{n+1} \leftarrow x_n + h \sum_{i=1}^{s} b_i k_i$
5. $z_n \leftarrow \sum_{i=1}^{s} l_i(0) Z_i$ with $l_i(t) = \prod_{j \neq i} \frac{t - c_j}{c_i - c_j}$
6. $M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[i]})$
7. compute $\frac{dK}{dw_n}$ using $M$ in (10)
8. $\partial x_{n+1}/\partial w_n \leftarrow \partial x_n/\partial w_n + h \sum_{i=1}^{s} b_i \frac{dk_i}{dw_n}$
9. $\partial z_n/\partial w_n \leftarrow \sum_{i=1}^{s} l_i(0) \frac{dz_i}{dw_n}$
10. compute outputs and sensitivities using $(K, \frac{dK}{dw_n})$

**IV. STRUCTURE EXPLOITING IRK METHODS**

This section starts by adapting the IRK methods to the specific three stage structure in Subsection IV-A, followed by some remarks in Subsection IV-B. The use of these methods from MATLAB in the ACADO Toolkit is briefly illustrated in Subsection IV-C.

**A. Implementation details**

The methods from Section III could be applied to the system in (2), ignoring the structure completely. This paper proposes an implementation which is equivalent to the latter, while exploiting the structure as much as possible. As illustrated in Figure 1, the scheme can be represented by four blocks in total including the one that handles the extra outputs and their sensitivities. The idea is to compute all the collocation variables $K$ and their derivatives $\frac{dK}{dw_n}$, so that the outputs and the sensitivities can still be generated in a similar way as before.
a) Linear input system: Let us define the collocation variables \( K_1 = (k_1^1, \ldots, k_n^1) \) for the linear input system. The resulting equations in (7) will also be linear, namely

\[
M_1 = \frac{\partial G_1}{\partial K_1} = \begin{pmatrix}
C_1 - ha_{11}A_1 & \cdots & -ha_{1s}A_1 \\
\vdots & \ddots & \vdots \\
-ha_{sl}A_1 & \cdots & C_1 - ha_{ss}A_1
\end{pmatrix} \quad (11)
\]

is a constant \( sN_1^1 \times sN_1^1 \) matrix in which \( G_1 \) defines the collocation equations. Instead of performing \( L \) Newton iterations like in (8), the variables can now be computed by \( K_1 = -M_1^{-1}G_1(x_0) \). The inverse matrix \( M_1^{-1} \) can be precomputed offline, which leaves us with only a matrix-vector multiplication. Even the derivatives \( \frac{dK_1}{dw_n} \) are constant and will therefore be precomputed.

b) Nonlinear system: For the implicit nonlinear system, the methods are still applied as described in Section III for the variables \( K_2 = (k_1^2, \ldots, k_N^2, Z_1, \ldots, Z_s) \) of dimension \( s(N_2^2 + N_z) \). The only difference is that the derivatives \( \frac{dK_2}{dw_n} \) are now needed to evaluate \( \frac{\partial G_2}{\partial w} \) for the sensitivity generation with \( G_2 \) defining the nonlinear collocation equations. In Figure 1, evaluations of the function \( f_j \) from (2c) and of its Jacobian are also considered to be part of this block so that it handles all nonlinear expressions.

c) Linear output system: Completely similar to the input system, the variables \( K_3 = (k_1^3, \ldots, k_N^3) \) can be computed by \( K_3 = -M_3^{-1}G_3(x_0) \), where \( M_3 = \frac{\partial G_3}{\partial x} \) is a \( sN_3^3 \times sN_3^3 \) matrix and \( G_3 \) defines the collocation equations for the linear output system. The inverse matrix \( M_3^{-1} \) and the derivatives of \( K_3 \) with respect to \( x_i^3 \) are constant and can be precomputed. The rest of the derivatives are computed by the matrix-vector multiplication \( \frac{dK_3}{dw_n} = -M_3^{-1} \frac{\partial G_3}{\partial w_n} \), using the derivatives \( \frac{dK_3}{dx_i^3} \) and \( \frac{dK_3}{dw_n} \) in the evaluation of \( \frac{\partial G_3}{\partial w_n} \).

B. Some small remarks

First of all, the Jacobian matrix \( \frac{dK}{dw_n} \) has a clear sparsity structure

\[
\frac{dK}{dw_n} = \begin{pmatrix}
\frac{dK_1}{dx_i^1} & 0 & 0 & \frac{dK_1}{dx_i^1} \\
\frac{dK_2}{dx_i^2} & \frac{dK_2}{dx_i^2} & 0 & \frac{dK_2}{dx_i^2} \\
\frac{dK_3}{dx_i^3} & \frac{dK_3}{dx_i^3} & \frac{dK_3}{dx_i^3} & \frac{dK_3}{dx_i^3}
\end{pmatrix}, \quad (12)
\]

which is exploited in the implementation of the integrators where possible. In addition, it is interesting to note that the in- and output system actually have an analytical solution since they consist of a set of linear differential equations with constant coefficients. Using this would however have little impact on the computational complexity and the overall accuracy of the integration method for the complete system.

C. The ACADO software

The presented integrators are part of the open source ACADO code generation tool [7]. It is implemented in C++ and efficient C-code is exported, tailored to a specific problem. A brief glance at this tool is made using its MATLAB interface. Consider a simple artificial system

\[
\begin{align*}
C_1 \dot{x}_1 &= A_1 x_1 + B_1 u, \\
x_2 &= a_T v_1, \\
C_2 \dot{x}_3 &= A_2 x_3 + x_2^2,
\end{align*} \quad (13)
\]

where the \( A, B \) and \( C \) matrices are constant, \( x_1, x_2 \) and \( x_3 \) are vectors of each 2 differential states, \( u \) is the control input and (13a)-(13c) correspond to the three stages from Section II. Let us now export a standalone integrator that exploits the structure in this model using ACADO from MATLAB:

```plaintext
1 DifferentialState x1(2) x2(2) x3(2);
2 Control u;
3 A1 = [ ]; A2 = [ ]; % 2x2 matrix
4 C1 = [ ]; C3 = [ ]; % 2x2 matrix
5 B1 = [ ]; % 2x1 vector
6 sim = acado.SIMexport( 0.1 );
7 sim.setLinearInput(C1, A1, B1);
8 sim.setModel(dot(x2) == [u*x1]);
9 sim.setLinearOutput(C3, A3, [x2.^2]);
10 sim.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL4' );
11 sim.set( 'NUM_INTEGRATOR_STEPS', 2 );
12 sim.exportCode('export')
```

Note that it is also possible to export these integrators directly as part of an NMPC algorithm.

V. APPLICATIONS IN OPTIMAL CONTROL

The goal of this section is both to illustrate the reasoning behind the proposed three stage structure and to present speedups that could be expected from the structure exploiting integrators. To strengthen this message, two real-world problems will be tackled. An application of NMPC on an overhead crane is discussed in Subsection V-A, following a Quadcopter example in Subsection V-B. The experiments presented in this section are performed using ACADO on an ordinary computer (Intel i7-3720QM 6MB cache, 2.60 GHz, 64-bit Ubuntu 12.04 and Clang 3.0).

A. NMPC of an overhead crane

A dynamic model very similar to the one in [15] is used for the same overhead crane. Let us immediately write down the equations in the presented three-stage format. The linear input system consists of

\[
\begin{align*}
\dot{x}_T &= v_T \\
\dot{v}_T &= a_T \\
\dot{u}_T &= u_T R
\end{align*}
\]

\[
\begin{align*}
\dot{x}_L &= v_L \\
\dot{v}_L &= a_L \\
\dot{u}_L &= u_L R
\end{align*}
\]

followed by two nonlinear equations:

\[
\begin{align*}
\hat{\theta} &= \omega \\
\omega &= \frac{1}{x_L} (g \sin(\theta) + a_T \cos(\theta) + 2v_L \omega)
\end{align*}
\]

where

\[
\begin{align*}
a_T &= \frac{1}{\tau_1} v_T + \frac{A_1}{\tau_1} u_T \\
a_L &= \frac{1}{\tau_2} v_L + \frac{A_2}{\tau_2} u_L
\end{align*}
\]
TABLE I
AVERAGE COMPUTATION TIMES FOR NMPC OF AN OVERHEAD CRANE.

<table>
<thead>
<tr>
<th></th>
<th>unstructured</th>
<th>structured</th>
</tr>
</thead>
<tbody>
<tr>
<td>integration method</td>
<td>220 µs</td>
<td>67 µs</td>
</tr>
<tr>
<td>condensing</td>
<td>6 µs</td>
<td>6 µs</td>
</tr>
<tr>
<td>QP solution (qpOASES)</td>
<td>16 µs</td>
<td>16 µs</td>
</tr>
<tr>
<td>remaining operations</td>
<td>3 µs</td>
<td>3 µs</td>
</tr>
<tr>
<td>one real-time iteration</td>
<td>245 µs</td>
<td>92 µs</td>
</tr>
</tbody>
</table>

Note that there is no linear output system and the different parameters are defined in [15]. The NMPC algorithm can now use an OCP formulation similar to (1) for e.g. a simple point-to-point motion. Table I illustrates the speedup for one iteration of the RTI scheme implemented in ACADO. A horizon of 1s and 10 control intervals is used with 4 integration steps of the 4th order Gauss method per interval. The table presents the average computation times for the different components with and without structure exploitation in the auto generated integrator. For this relatively small model, a speedup factor of 3 can already be observed for the total computation time spent in integration and sensitivity generation. Note that no extra constraint has been used, while in reality there are at least box constraints defined on the control inputs. In the used SQP framework, these will be handled by the online QP solver qpOASES.

It has been illustrated that a linear input system can appear naturally. Consider now the situation where it would be useful to penalize the high frequency content in the states. One way to do this is by implementing a High-Pass RC filter such as in (4) for each of the states. This results in the following 6 equations being added to the input system:

\[
x^{HP}_L = x_L - \hat{\omega}_C x^{HP}_L,
\]

\[
y^{HP}_L = y_L - \hat{\omega}_C y^{HP}_L,
\]

\[
\dot{\theta}^{HP} = \hat{\theta} - \hat{\omega}_C \theta^{HP},
\]

\[
\dot{\omega}^{HP} = \hat{\omega} - \hat{\omega}_C \omega^{HP}.
\]

The new states in these equations are modeling the high frequency content in the corresponding states from the original system, leading to a complete system of 16 differential states. Weighting these high frequency states in the NMPC formulation causes a certain smoothing of the closed-loop trajectories, as depicted in Figure 2. Similar to before, Table II now presents a speedup factor of 6 for the integration time due to the structure exploitation on this extended model.

B. NMPC of a Quadcopter

In this second example, NMPC will be applied to a Quadcopter (see e.g. [16]). This paper uses a simple Quadcopter model, where it is assumed that low-level speed controllers ensure the tracking of the reference velocities of the propellers, which are computed by the NMPC scheme.

The model equations are briefly summarized by

\[
\ddot{\omega}_k = \tau_k - \dot{\omega}_k = \tau_k - \omega_k,
\]

\[
\dot{\theta} = \frac{1}{2} \sum_{k=1}^{4} \rho AC L \omega_k^2 \mathbf{1}_z, \quad T = \sum_{k=1}^{4} \frac{1}{2} \rho AC D \omega_k^2 \mathbf{1}_z,
\]

\[
\ddot{p} = \nu, \quad \dot{l}_p = p - \nu_k,
\]

where \(U_k\) for \(k = 1, \ldots, 4\) are the control inputs, commanding the 2nd time derivative of the propeller reference velocities \(\omega_k\), \(\omega_k\), the actual propeller velocities, \(q \in \mathbb{R}^4\) is the quaternion vector used to represent the orientation, \(\Omega\) is the main body angular velocity in the body frame, \(v\) is the linear velocity in the inertial frame, \(p\) is the position and \(l_p \in \mathbb{R}^3\) the integral of the position error. Matrix \(R = EG^T\) is the rotation matrix between the body frame and the inertial frame, with

\[
G = \begin{bmatrix}
-q_1 & -q_2 & -q_3 \\
q_0 & -q_3 & q_2 \\
q_3 & q_0 & -q_1 \\
-q_2 & q_1 & q_0
\end{bmatrix},
\]

\[
E = \begin{bmatrix}
-q_1 & -q_2 & -q_3 \\
q_0 & q_3 & q_2 \\
-q_3 & q_0 & q_1 \\
q_2 & -q_1 & q_0
\end{bmatrix}.
\]

Parameter \(\tau\) is the time constant of the speed control loops, \(J \in \mathbb{R}^{3 \times 3}\) is the inertia matrix of the Quadcopter, and \(m\) its total mass. Coefficients \(C_L\) and \(C_D\) are respectively the lift and drag coefficients, \(A\) is the individual area of the propellers, \(\rho\) is the air density, and \(\mathbf{1}_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T\).
Fig. 3. Illustration of a closed-loop trajectory of the position and orientation of the Quadcopter.

### Table III

<table>
<thead>
<tr>
<th>Component</th>
<th>Unstructured</th>
<th>Structured</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration method</td>
<td>35.8 ms</td>
<td>3.1 ms</td>
</tr>
<tr>
<td>Condensing</td>
<td>2.6 ms</td>
<td>2.6 ms</td>
</tr>
<tr>
<td>QP solution (qpOASES)</td>
<td>0.1 ms</td>
<td>0.1 ms</td>
</tr>
<tr>
<td>Remaining operations</td>
<td>0.2 ms</td>
<td>0.2 ms</td>
</tr>
<tr>
<td>One real-time iteration</td>
<td>38.7 ms</td>
<td>6.0 ms</td>
</tr>
</tbody>
</table>

It can be observed that (19) forms a 12 states linear input system, (20)-(21) forms a 10 states nonlinear dynamic system and (23) is a 6 states linear output system. The input dynamics $\omega_k^{\text{ref}} = U_k$ are used to implement a penalty on the high frequency components in the control input, i.e. the Lagrange term

$$\Pi_{\text{input}} = \sum_{k=1}^{4} W_1 U_k^2 + W_2 (\omega_k^{\text{ref}})^2$$

is added to the objective, with positive weights $W_1$ and $W_2$. Moreover, the integral of the position error $I_p = p - p_{\text{ref}}$ is introduced and penalized in order to eliminate steady-state errors in the Quadcopter position, which typically result from constant disturbances or model errors.

The positive definite weighting matrices $Q$, $R$ and $P$ are chosen to achieve quick point-to-point motions. Again no extra constraints were mentioned here for brevity, although it fits the OCP formulation in (1). A possible closed-loop trajectory for the position and orientation of the Quadcopter is presented in Figure 3, in the event of a motion from point $(1,1,1)$ to the origin. Table III shows a speedup factor of 12 for the total integration time. For these experiments, a horizon of length 5s with 25 control intervals has been used and 2 integration steps of the 6th order Gauss method are performed per interval.

### VI. Conclusions & Further Developments

This paper proposed a new format for defining nonlinear dynamic models and showed how the three-stage structure therein can be strongly exploited by IRK methods. The general idea of code generation for IRK methods with efficient generation of sensitivities and continuous output was first briefly repeated. The mentioned structure in the differential equations has then been introduced and motivated, followed by a discussion on the consequences for the implementation when exploiting this. Two problems, namely the real-time control of an overhead crane and of a Quadcopter, illustrated the relevance of the proposed three-stage structure. Numerical experiments have shown that important speedups can be achieved with these auto generated, structure exploiting integrators which are available in the ACADO toolkit [17].

It is useful to note that the proposed three-stage structure could be detected and exploited automatically in any nonlinear model. Future work could include implementing such an automatic structure detection.

### References