Fast auto generated ACADO integrators and application to MHE with multi-rate measurements

Rien Quirynen, Sébastien Gros and Moritz Diehl

Abstract— **Algorithms for real-time, embedded optimization need to run within tight computational times, and preferably on embedded control hardware for which only limited computational power and memory is available. A computationally demanding step of these algorithms is the model simulation with sensitivity generation. This paper presents an implementation of code generation for Implicit Runge-Kutta (IRK) methods with efficient sensitivity generation, which outperforms other solvers for the targeted applications. The focus of this paper will be on the extension of the proposed tool to the integration of index-1 Differential Algebraic Equations (DAE), and continuous output functions, which are crucial for e.g. performing sensor fusion with measurements provided at very high sampling rates. The new tool is provided with a powerful MATLAB interface. It is illustrated in simulation for the trajectory estimation of a mechanical system modeled by complex Differential-Algebraic equations, using sensor information provided at fast, multi-rate sampling frequencies.**

Keywords **: code generation, IRK methods, sensitivity generation, continuous output, MHE, multi-rate measurements**

I. INTRODUCTION

Moving Horizon Estimation (MHE) [1] and Nonlinear Model Predictive Control (NMPC) [2], [3] are popular for their ability to explicitly handle constraints and nonlinear dynamics. For fast systems, the high computational burden of MHE and NMPC is however a major challenge. Both MHE and NMPC involve solving an optimization problem at each sampling time, while respecting the time limitation imposed by the real-time implementation. Recent algorithmic progresses [4], [5], [6] allow for considering MHE and NMPC for very fast systems. Among the various available algorithms, the Real-Time Iteration (RTI) scheme [7] has been proposed as a highly competitive approach to fast MHE and NMPC.

The dominating computational component of these algorithms is the simulation of the model equations, with sensitivity generation. For real-time optimization, in order to guarantee a deterministic runtime, an integration method with fixed order and step size is preferable. In a multiple-shooting framework, integration takes place over relatively small time intervals. In such a context, single-step methods such as Runge-Kutta (RK) schemes are especially attractive, because they do not need a start-up procedure. Moreover, the Implicit RK (IRK) methods allow for a natural extension to Differential Algebraic Equations (DAE). An efficient implementation in combination with sensitivity generation has already been presented in [8].

Collocation methods form a specific group of IRK methods with the capability of providing outputs between grid points, at the cost of a single polynomial evaluation. In the context of MHE, continuous-output integrators allow for using a larger integration step size than the actual frequency of the measurements, such that high-frequency sensor information can be included in the estimation scheme at a very low computational cost.

In addition to using efficient algorithms, code generation allows for a significant improvement of the computational time by removing from the code the unnecessary computations, by optimizing memory accesses, cache usage, and by efficiently exploiting the problem dimensions and sparsity patterns. Code generation is e.g. used in the ACADO toolkit, where highly efficient, self-contained C-code can be automatically exported [9]. The IRK method presented in this paper is provided as a component of ACADO toolkit, can be code-generated, and is also available in MATLAB.

The paper is organized as follows. Section II describes the implementation of auto generated IRK methods for index-1 DAE systems, including a discussion on sensitivity generation and continuous output. A user friendly MATLAB interface is then presented in Section III with a tutorial use case. Section IV then illustrates the performance of the integrators on an estimation problem with a focus on the continuous output feature and the corresponding sensitivity generation.

Contribution: This paper presents an auto-generated IRK method for the integration of index-1 DAE systems with efficient sensitivity generation, and the ability to compute outputs on an arbitrary time grid at a very low additional computational cost, hence allowing for considering new applications such as e.g. MHE with multi-rate, high frequency sensor information.

R. Quirynen, S. Gros and M. Diehl are with the Optimization in Engineering Center (OPTEC), KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium. rien.quirynen@esat.kuleuven.be

* This research was supported by Research Council KUL: PFV/10/002 Optimization in Engineering Center OPTEC, GOA/10/09 MaNet and GOA/10/11 Global real- time optimal control of autonomous robots and mechatronic systems. Flemish Government: IOF/KP/SCORES4CHEM, FWO: PhD/postdoc grants and projects: G.0320.08 (convex MPC), G.0377.09 (Mechatronics MPC); IWT: PhD Grants, projects: SBO LeCoPro; Belgian Federal Science Policy Office: IUAP P7 (DYSCO, Dynamical systems, control and optimization, 2012-2017); EU: FP7-EMBOCON (ICT-248940), FP7-SADCO (MC ITN-264735), ERC ST HIGHWIND (259 166), Eurostars SMART, ACCM. The first author holds a PhD fellowship of the Research Foundation – Flanders (FWO).

II. Auto generated IRK methods

This section presents code generation for IRK methods with efficient sensitivity generation. The formulation of these methods for ODE and DAE systems of index 1 is treated in Subsection II-A. Some implementation details are mentioned in Subsection II-B and the efficient computation of sensitivities is discussed in Subsection II-C. Subsection II-D highlights an important extra feature of these auto generated integrators, namely the continuous output.

A. Formulation

The assumption in this paper is that the following Initial Value Problem (IVP) needs to be solved over a certain time interval:

$$\begin{aligned} 0 &= f(t,\dot{x}(t),x(t),z(t),p(t)), \\ x(0) &= x_0, \end{aligned} \tag{1}$$

with $x(t)$ a vector of N_x differential states and $\dot{x}(t)$ the corresponding time derivatives, $z(t)$ a vector of N_z algebraic states, $p(t)$ a vector of parameters and f defines a system of N_x+N_z equations. The presented integrators are therefore able to handle all models ranging from explicit ODE to implicit DAE systems. Note that this DAE formulation is more general than the one used in [8]. The only assumption here is that the DAE system is of index 1. The algebraic states z and the differential state derivatives $\dot{x}$ are uniquely defined by the parameters and differential states, i.e. the matrix $\frac{\partial f}{\partial(z,\dot{x})}$ is invertible [10].

An s-stage RK method is defined by the coefficients c_i, the weights b_i and the internal coefficients a_{ij}, $\forall i,j = 1,\ldots,s$. The method is explicit if the matrix A is strictly lower triangular, otherwise it is implicit. Latter methods involve more computational effort, but they generally also provide a higher order of accuracy and better stability properties. The focus of this paper will be on A-stable IRK methods such as the Gauss or Radau methods [11]. Runge-Kutta methods are often represented using a Butcher table:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{2}$$

An s-stage IRK method can be applied to the system in (1) for the integration from t_n until $t_{n+1} = t_n + h$, resulting in:

$$\begin{aligned} 0 &= f(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij}k_j, Z_i),\ i = 1,\ldots,s, \\ x_{n+1} &= x_n + h\sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{3}$$

with Z_i the stage values of the algebraic states z and k_i the stage values for $\dot{x}$. This nonlinear system of $(N_x+N_z)\times s$ equations can then be solved using a variant of the Newton method. Different from the approach in [8], consistency of the state values is here only assured at the s collocation nodes. One reason for this is the associated computational cost in case of the general system in (1). Another reason is that stiffly accurate IRK methods such as the Radau IIA methods already include the endpoint as one of these nodes.

B. Implementation

Solving the nonlinear system in (3) will be done using a variant of the Newton method which always performs a fixed amount of L iterations. The presented integrators also use a fixed step size and a fixed order. The reason for this is that the ACADO code generation tool is mainly used for real-time applications. The resulting need for a deterministic computation time has a big impact on the implemented algorithms. Given a good initialization of the variables $K = (k_1,\ldots,k_s,Z_1,\ldots,Z_s)$, a few iterations are sufficient in practice [8]. Since successive values of the Jacobian do not differ much in this case, only one evaluation is done per integration step. This will be discussed further in Subsection II-C.

Let us write the nonlinear system as $G(x_n,K) = 0$. The Newton method can then be summarized as

$$\begin{aligned} M &= \frac{\partial G}{\partial K}(x_n,K^{[0]}) \\ K^{[i]} &= K^{[i-1]} - M^{-1}G(x_n,K^{[i-1]}),\ i = 1,\ldots,L \end{aligned} \tag{4}$$

The auto generated code therefore consists of two major components, namely the evaluation of the Jacobian and the solution of a linear system. The Jacobian $\frac{\partial G}{\partial K}(x_n,K)$ is a $s(N_x+N_z)\times s(N_x+N_z)$ matrix that has the following structure:

$$\begin{pmatrix} \frac{\partial f_1}{\partial \dot{x}} + ha_{11}\frac{\partial f_1}{\partial x} & \cdots & ha_{1s}\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial z} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ ha_{s1}\frac{\partial f_s}{\partial x} & \cdots & \frac{\partial f_s}{\partial \dot{x}} + ha_{ss}\frac{\partial f_s}{\partial x} & 0 & \cdots & \frac{\partial f_s}{\partial z} \end{pmatrix}, \tag{5}$$

where $f_i = f(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij}k_j, Z_i)$. Evaluating the different derivatives of the function f is done using Algorithmic Differentiation (AD) [12]. According to the discussion in [8], the linear system $M\Delta K^{[i]} = -G(x_n,K^{[i-1]})$ is solved using a LU decomposition of the matrix M. This factorization can then be reused to solve the linear systems with only a different right-hand side. It is important to note the sparsity structure in (5), which should be exploited for maximum efficiency.

C. Sensitivity generation

In addition to the state values at the next time point, the generated integrators can provide accurate sensitivities in an efficient way. This is for example necessary in the context of dynamic optimization. Let us assume that $w_n = (x_n,p)$ denotes all the independent variables with respect to which the first order derivatives are needed. The goal is then to efficiently compute the matrix $\partial(x_{n+1},z_{n+1})/\partial w_n$. Since the continuous output is an important feature of the presented integrators, only forward propagation of the sensitivity information is considered.

A thorough discussion on forward techniques of sensitivity generation for IRK methods can be found in [13]. But let us emphasize that the use of the Variational Differential

Algorithm 1 One step of the IFT-R implementation

Input: w_n, initial $K^{[0]}$, LU factorization of M
Output: $(x,z)_{n+1}$ and $\partial(x,z)_{n+1}/\partial w_n$
1: **for** $i = 1 \rightarrow L$ **do**
2: $\quad K^{[i]} \leftarrow K^{[i-1]} - M^{-1}G(w_n, K^{[i-1]})$
3: **end for**
4: $x_{n+1} \leftarrow x_n + h\sum_{i=1}^{s} b_i k_i$
5: $z_{n+1} \leftarrow \sum_{i=1}^{s} l_i(1) Z_i \quad$ with $\; l_i(t) = \prod_{j \neq i} \frac{t-c_j}{c_i-c_j}$
6: $M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[L]})$
7: factorize M
8: compute sensitivities using M in (6)
9: initialize $K^{[0]}$ for next integration step

Algebraic Equations (VDAE) is not suited for an implicit integration method. Using finite differences can lead to relatively inaccurate results. And the principle of Internal Numerical Differentiation (IND) from [14] results in an iterative scheme, which is costly when many directional derivatives are needed. A better approach is therefore to abuse the fact that an approximation of the Jacobian $\partial G/\partial K$ is needed for the Newton iterations in (4) anyway. This means that evaluating the Jacobian to compute sensitivities makes sense if it can be reused in the Newton iterations of the next integration step. Applying the Implicit Function Theorem (IFT) to $G(w_n, K) = 0$, results in

$$\frac{dK}{dw_n} = -\frac{\partial G}{\partial K}^{-1} \frac{\partial G}{\partial w_n}. \tag{6}$$

These sensitivities of the variables K can then be used to obtain the sensitivities $\partial(x_{n+1}, z_{n+1})/\partial w_n$. It is a direct approach that can compute the sensitivities up to machine precision. In the IFT-R implementation, the Jacobian is reused in the next integration resulting in only one factorization per step [8]. The latter approach is described by Algorithm 1.

D. Continuous output

Some promising possibilities of auto generated IRK methods with continuous output have already been mentioned in the introduction and their use will also be illustrated in Section IV. To make this possible, a continuous extension of the methods is needed. Let us however focus on the family of collocation methods, which already provide a continuous interpolation. For a collocation method, the coefficients c_i need to be distinct and they completely define the coefficients b_i and a_{ij}. This way, a polynomial $p(t)$ can be constructed that passes through (t_n, x_n) and that agrees with the model in (1) at the s corresponding nodes. Evaluating this polynomial at time $t_n + ch$ results in

$$x(t_n + ch) \approx x_n + h\sum_{j=1}^{s} k_j \int_0^c l_j(\tau)\,\mathrm{d}\tau, \tag{7}$$

where $l_i(t) = \prod_{j \neq i} \frac{t-c_j}{c_i-c_j}$ are the Lagrange interpolating polynomials. Note that for c equal to 1, this corresponds to the computation of x_{n+1} in (3). In the case of an s-stage IRK method of order p, the order of this continuous output is $p^* = \min(p, s+1)$ [15].

It is now possible to do something equivalent for the algebraic states. However, this would mean that consistent values of the states at time t_n are needed which is not evident. In the case of a discontinuous jump in the value of a parameter or control input, the differential states will namely vary continuously while the algebraic states could also exhibit such a jump [16]. Extra computation time spent to achieve consistency at the beginning of the integration step is undesirable. The interpolating polynomial for the algebraic states is therefore one order less and similar to the one for the differential state derivatives:

$$\begin{aligned} z(t_n + ch) &\approx \sum_{i=1}^{s} l_i(c) Z_i, \\ \dot{x}(t_n + ch) &\approx \sum_{i=1}^{s} l_i(c) k_i. \end{aligned} \tag{8}$$

The conclusion from this should be that the differential as well as the algebraic states can efficiently be computed on a grid finer than the integration grid. The auto generated integrators exploit this by allowing the user to also define one or more output functions like

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

with a corresponding grid on which they should be evaluated. Using the computed derivatives of the variables K from (6), also sensitivities of these extra outputs can be obtained. This is important when one would like to use these outputs in an optimization problem like in Section IV. Algorithms describing the computation of extra outputs and of their sensitivities can be found in [13].

III. The MATLAB interface

The presented integrators have been implemented as an add-on to the ACADO Toolkit and are part of its code generation tool [17], [18]. It is implemented in C++ and the exported C-code is self-contained and efficient, since it is tailored to the specific problem formulation. The motivation behind a MATLAB interface to this code generation tool for integrators is discussed in Subsection III-A and a tutorial use case is presented in Subsection III-B.

A. The motivation

The goal of the MATLAB interface to ACADO is first of all to lower the threshold for potential users of advanced dynamic optimization algorithms who are more familiar with MATLAB than with C++. It is also interesting because of all the other features and algorithms that are available in the MATLAB environment, resulting in interesting prototyping possibilities and user-friendliness. A limited MATLAB interface already existed and is described in the ACADO for MATLAB User's Manual [19].

The motivation for a MATLAB interface is stronger in case of code generation, since high-level access to the efficient, auto generated C-code could be even more valuable [20]. An

interface to important functions in the C-code can be automatically generated in the form of MEX-files. This avoids the need for any knowledge of C/C++ and allows the user to formulate the problem as well as to run simulations from MATLAB, while keeping the performance of the optimized C-code.

B. A tutorial use case

The considered test problem is a small nonlinear ODE model for an overhead crane, similar to the one used in [18]. The system has 8 differential states and 2 control inputs. It is possible to export the presented integrators within NMPC or MHE, as illustrated in [8]. Let us however focus on the possibility to export them as standalone components which can then be used outside of the ACADO environment. The user's goal here would be to have an idea about the performance of different methods and the corresponding step size to be used for this model.

a) Formulating the model: After defining the states and control inputs, the expressions of the ODE system can be constructed as follows:

```
DifferentialState xT vT xL vL phi omega uT uL;
Control duT duL;

h    = 0.01;
tau1 = 0.0128;  a1  = 0.0474;
tau2 = 0.0247;  a2  = 0.0341;
g    = 9.81     m   = 1318;

aT   = -1.0/tau1*vT + a1/tau1*uT;
aL   = -1.0/tau2*vL + a2/tau2*uL;

f = [ vT; aT; vL; aL; omega; ...
      1.0/xL*(-g*sin(phi)-aT*cos(phi)- ...
      2*vL*omega); duT; duL ];

sim = acado.SIMexport( h );
sim.setModel(f);
```

In accordance with the discussion of Subsection II-D, it is also possible to define one or more output functions with their corresponding amount of measurements per interval. An example of this is the following:

```
sim.addOutput([xT; vT; xL; vL]);
sim.setMeasurements(100);
```

b) Specify the integrator: The step size should be chosen as well as the specific integration method to be exported. Examples of available IRK schemes are the Radau IIA methods of order 1, 3 and 5 and the Gauss methods of order 2, 4, 6 and 8. Also explicit RK schemes are available with orders ranging from 1 until 4. Other options for the code generation can be specified. Let us export the Gauss method of order 4 with only one step of size $h = 0.01$:

```
sim.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL4' );
sim.set( 'NUM_INTEGRATOR_STEPS', 1        );
```

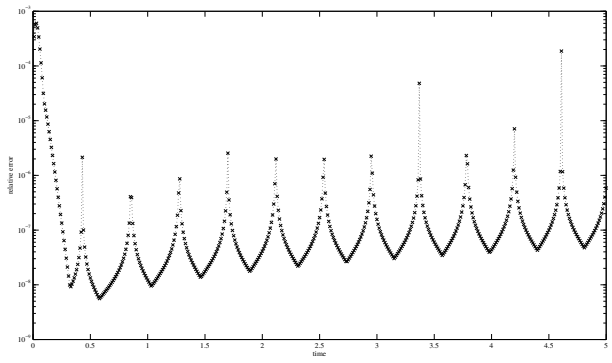


Fig. 1. The resulting figure from the tutorial use case, showing the maximum relative error of the simulated states over 5s.

c) The exported code: It is now possible to automatically generate and run a test, which checks the performance of the integrator by comparing it with the results for a much smaller step size. It is however important to emphasize that it is possible to let the tool only export the code, together with a MEX-function to call the integrator and one to call only the right-hand side of the model. The latter can be used with one of the standard MATLAB integrators such as ode45 or ode15s. This provides the user with the tools to easily use the solver, while keeping the performance of the optimized C-code. It is for example possible to do some simulations and compare the results with those of an integrator available in MATLAB:

```
sim.set( 'GENERATE_MATLAB_INTERFACE', 1 );
sim.exportCode;

test_RUN
mex integrate.c integrator.c
mex rhs.c integrator.c

x  = rand(8,1); u = zeros(2,1);
xs = x;         N = 500;
for i = 1:N
    [states out] = integrate(xs(:,end),u);
    xs(:,end+1)  = states.value;
end

options=odeset('RelTol',1e-12,'AbsTol',1e-12);
[tout exact] = ode15s(@(t, y) rhs(t, y, u), ...
                      [0:h:N*h],x,options);
exact = exact';
err = max(abs(xs-exact)./abs(exact),[],1);
semilogy([h:h:N*h], err(2:end), ':kx');
```

The eventual results of this tutorial when running the MATLAB code are shown in Figure 1.

IV. Application: MHE with multi-rate measurements

This section presents an example using the integrators for MHE with multi-rate measurements. Figure 2 depicts the set-up that is considered. The system consists of a cube hanging to a rod by one of its corners. The model is developed using multi-body modeling and the rotation-less

formulation, where the position of the center of mass of the cube $p = [x, y, z]$ and the orientation of the cube given by the rotation matrix $R \in \mathbb{R}^{3\times3}$ are the generalized coordinates. In this framework, the kinetic and potential energy of the cube are given by:

$$T = \frac{1}{2} m \dot{p}^T \dot{p} + \frac{1}{2} \omega^T J \omega, \quad V = mgz$$

where m is the mass of the cube, $J \in \mathbb{S}^3$ is the inertia tensor of the cube in its reference frame and $\omega \in \mathbb{R}^3$ is the angular velocity of the cube in its own reference frame. The motion of the cube is constrained by the rod to evolve on the manifold:

$$c(p,R) = \frac{1}{2}\left(p_c^T p_c - L^2\right) = 0, \quad p_c = p + RC$$

where $C = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$ is the position of the attached corner in the cube reference frame, $p_c \in \mathbb{R}^3$ is the position of the attached corner in the fixed reference frame, and $L = 1$ is the length of the rod.

The Lagrangian of the cube then reads:

$$\mathcal{L} = T - V - \nu c(p,R) - \operatorname{tr}\left(Z\left(R^T R - I\right)\right)$$

where $\nu \in \mathbb{R}$ is the Lagrange multiplier associated to the constraint c, and $Z \in \mathbb{S}^3$ is the symmetric matrix of Lagrange multipliers associated to the orthonormality of the rotation matrix $R \in \mathcal{SO}(3)$. Defining the linear operator $P : \mathbb{R}^{3\times3} \times \mathbb{R}^{3\times3} \to \mathbb{R}^3$, $P(A,R) = U(R^T A)$ where

$$U\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} a_{32} - a_{23} \\ a_{13} - a_{31} \\ a_{21} - a_{12} \end{bmatrix},$$

it can be shown [21] that the dynamics of the cube then reduce to the following index-1 DAEs:

$$\dot{R} = R\omega$$

$$\begin{bmatrix} J & 0 & \nabla_p c \\ 0 & J & 2P(\nabla_R c) \\ \nabla_p c^T & 2P(\nabla_R c)^T & 0 \end{bmatrix} \begin{bmatrix} \ddot{p} \\ \dot{\omega} \\ \nu \end{bmatrix} = \begin{bmatrix} -\nabla_p V \\ -\omega \times J\omega \\ -\operatorname{tr}\left(\nabla_R \dot{c}^T R\omega\right) + \nabla_p \dot{c}^T \dot{p} \end{bmatrix},$$

with the consistency conditions:

$$\begin{aligned} & c = 0 \\ & \dot{c} = \left(\operatorname{tr}\left(\nabla_R c^T R\omega\right) + \nabla_p c^T \dot{p}\right)_{t=t_0} = 0 \\ & \left(R^T R\right) = I \end{aligned} \tag{10}$$

that must be enforced at any time t_0 of the trajectory. In the following, (10) will be lumped into $H = \begin{bmatrix} c & \dot{c} & R^T R - I \end{bmatrix}$.

The goal here is to accurately estimate the position and orientation of the moving cube, using two different types of measurements [22]. Absolute location information is available at a low frequency of 10Hz using the Global Positioning System (GPS). In addition, measurements of the angular velocity and linear acceleration are provided by an Inertial Measurement Unit (IMU) at a high frequency of 800Hz. This

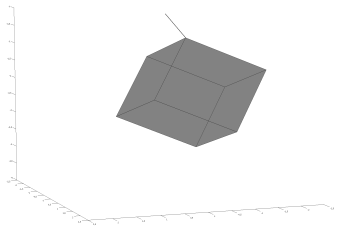


Fig. 2. The set-up used in the estimation problem of Section IV.

estimation problem can be handled using MHE, based on a moving horizon formulation. An optimal estimate x_N for the states of the system is then obtained by repeatedly solving the following dynamic optimization problem:

$$\begin{aligned} \underset{x_0,\ldots,x_N}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \frac{1}{2} \|R_k(x_k)\|_2^2 \\ \text{subject to} \quad & x_{k+1} = G_k(x_k), \quad \forall k = 0,\ldots,N-1, \\ & 0 = H(x_N), \end{aligned} \tag{11}$$

where R_k contains the difference between the real measurements and the outputs from the integrator and G_k corresponds to the integration from one GPS sample to the next. There are also 8 extra equality constraints that need to be satisfied. We impose them at the end point $H(x_N) = 0$ and the dynamics of the system will make sure they are satisfied over the whole horizon.

The result is an optimization problem with a least squares objective and only equality constraints. Let us assume that the number of estimation intervals $N = 10$ and the horizon is 1s. This OCP can be solved at each time point using multiple shooting in combination with the Constrained Gauss-Newton (CGN) method from [23]. Using the new integrators and corresponding interface, this algorithm can be implemented in an easy but still efficient way in MATLAB. The simulated values corresponding to the high frequency IMU measurements can be obtained using the continuous output feature as previously presented in [24]. This allows us to use an integration step size that is significantly larger than the time between two IMU measurements without any loss of information. In addition to these values, their sensitivities are also needed for the linearization done by the CGN method.

Figure 3 shows some results of a simulation over 2s. The simulated values of the linear velocity of the cube are presented as well as the estimates from the MHE algorithm. To the GPS and IMU measurements that are used in this simulation, Gaussian noise has been added with a standard deviation equal to 5% of that of the signal. It is clear from the figure that the MHE algorithm is still estimating the states very well. It is of course possible to implement the same algorithm using a standard DAE solver from MATLAB such as ode15s. The sensitivities of the outputs can then be obtained approximately using finite differences. A comparison between both implementations is given in Table I. The

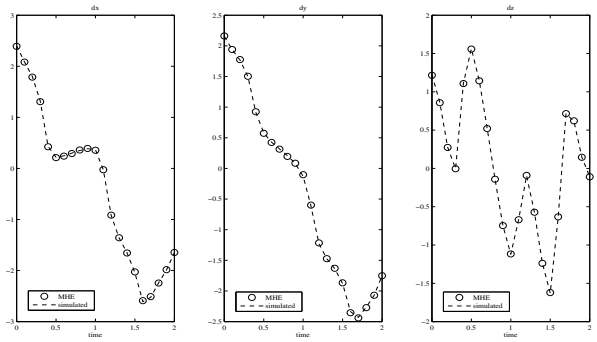


Fig. 3. The linear velocity states from a simulation of the system over 2s, together with the MHE algorithm estimating all the states at a sampling time of 0.1s.

	ACADO	ode15s
integrator calls	10	190
integration + sensitivities	0.02 s	10.65 s
total computation time	0.04 s	10.69 s
ratio spent in integrator	50 %	99 %

TABLE I

COMPARISON OF THE COMPUTATION TIME FOR ONE CGN ITERATION, USING RESPECTIVELY AN AUTO GENERATED METHOD FROM ACADO AND MATLAB'S DAE SOLVER ODE15S.

numerical experiments are run on an ordinary computer (Intel P8600 3MB cache, 2.40 GHz, 64-bit Ubuntu 12.04 LTS and MATLAB R2011a 64 bit) and the time measurements are done using the `tic`/`toc` functions in MATLAB.

A Radau IIA method of order 3 and fixed step size 0.02s is exported using ACADO, while MATLAB's ode15s implements a variable-order, variable-step method. The ACADO integrators can provide the sensitivities in an accurate and efficient way while ode15s needs to be called 19 times (18 states in the system) to obtain relatively inaccurate sensitivities. The accuracy of the sensitivities will strongly influence the convergence behavior of the algorithm, i.e. the amount of iterations. Table I therefore compares the cost of one iteration for the two different implementations. Keep in mind that 10 estimation intervals are being used. It should be clear that the auto generated integrators lead to a much faster algorithm and they also provide more accurate sensitivities which is very important in optimization.

While [8] showed that the ACADO auto generated methods can be approximately 100 times faster than the solvers from SUNDIALS, a speedup of factor 500 is observed here with respect to ode15s. The important conclusion here should not be that the auto generated methods are faster than a standard MATLAB integrator, since that is what would be expected. The complete message is that they are much faster, equally easy available in MATLAB and more powerful because of the sensitivity generation and the continuous output feature.

V. CONCLUSION & FURTHER DEVELOPMENTS

This paper has proposed code generation for IRK methods with efficient computation of sensitivities and a powerful continuous output feature. A compact discussion on the formulation and implementation of IRK methods for index-1 DAE systems has been given. The implementation of sensitivity generation and of the continuous output has been discussed. Also a MATLAB interface to this code generation tool for integrators has been presented and a nontrivial multi-rate MHE application has been used to show its power and usefulness. Numerical experiments confirmed that auto generated IRK methods deliver a much better performance than other solvers for the targeted applications. It is eventually important to note that the ACADO integrators are available as open source code [25].

Future work will expand the proposed methods by improving the support for larger problems, exploiting the sparsity present in the linear systems and by implementing code generation for other integration methods. Also the MATLAB interface will be improved and maintained.

REFERENCES

[1] C. Rao and J. Rawlings, "Nonlinear Moving Horizon State Estimation," in *Nonlinear Predictive Control* (F. Allgöwer and A. Zheng, eds.), (Basel Boston Berlin), pp. 45–69, Birkhäuser, 2000.

[2] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: stability and optimality," *Automatica*, vol. 26, no. 6, pp. 789–814, 2000.

[3] L. Simon, Z. Nagy, and K. Hungerbuehler, *Nonlinear Model Predictive Control*, vol. 384 of *Lecture Notes in Control and Information Sciences*, ch. Swelling Constrained Control of an Industrial Batch Reactor Using a Dedicated NMPC Environment: OptCon, pp. 531–539. Springer, 2009.

[4] C. Jones and M. Morari, "Polytopic approximation of explicit model predictive controllers," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2542–2553, 2010.

[5] C. Kirches, L. Wirsching, S. Sager, and H. Bock, "Efficient numerics for nonlinear model predictive control," in *Recent Advances in Optimization and its Applications in Engineering* (M. Diehl, F. F. Glineur, and E. J. W. Michiels, eds.), pp. 339–357, Springer, 2010.

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

[7] M. Diehl, H. Bock, and J. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[8] R. Quirynen, M. Vukov, and M. Diehl, "Auto Generation of Implicit Integrators for Embedded NMPC with Microsecond Sampling Times," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands* (A. F. Lazar, Mircea, ed.), vol. 4, 2012.

[9] H. Ferreau, *Model Predictive Control Algorithms for Applications with Millisecond Timescales*. PhD thesis, K.U. Leuven, 2011.

[10] R. Findeisen and F. Allgöwer, "Nonlinear model predictive control for index–one DAE systems," in *Nonlinear Predictive Control* (F. Allgöwer and A. Zheng, eds.), (Basel Boston Berlin), pp. 145–162, Birkhäuser, 2000.

[11] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*. Berlin Heidelberg: Springer, 2nd ed., 1991.

[12] A. Griewank and A. Walther, *Evaluating Derivatives*. SIAM, 2 ed., 2008.

[13] R. Quirynen, "Automatic code generation of Implicit Runge-Kutta integrators with continuous output for fast embedded optimization," Master's thesis, KU Leuven, 2012.

[14] H. Bock, "Recent advances in parameter identification techniques for ODE," in *Numerical Treatment of Inverse Problems in Differential and Integral Equations* (P. Deuflhard and E. Hairer, eds.), Boston: Birkhäuser, 1983.

[15] N. H. Cong and L. N. Xuan, "*Parallel-Iterated RK-type PC Methods with Continuous Output Formulas*," *International Journal of Computer Mathematics*, vol. 80:8, pp. 1025–1035, 2003.

[16] R. Hannemann, *Adjoint Sensitivity Analysis: Theory, Numerical Methods and Applications from Chemical Engineering*. PhD thesis, AVT - Aachener Verfahrenstechnik, 2011.

[17] H. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl, "High-speed moving horizon estimation based on automatic code generation," in *Proceedings of the 51th IEEE Conference on Decision and Control (CDC 2012), Hawaii*, 2012. (accepted).

[18] M. Vukov, W. V. Loock, B. Houska, H. Ferreau, J. Swevers, and M. Diehl, "Experimental Validation of Nonlinear MPC on an Overhead Crane using Automatic Code Generation," in *The 2012 American Control Conference, Montreal, Canada.*, 2012.

[19] D. A. et al., ACADO for Matlab User's Manual. OPTEC, 2010.

[20] B. Houska, H. Ferreau, and M. Diehl, "An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

[21] S. Gros, M. Zanon, M. Vukov, and M. Diehl, "Nonlinear MPC and MHE for Mechanical Multi-Body Systems with Application to Fast Tethered Airplanes," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, 2012.

[22] K. Geebelen, A. Wagner, S. Gros, J. Swevers, and M. Diehl, "Moving Horizon Estimation with a Huber Penalty Function for Robust Pose Estimation of Tethered Airplanes," in *Proceedings of the 51th IEEE Conference on Decision and Control (CDC 2012), Hawaii*, 2012.

[23] H. Bock, *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, vol. 183 of *Bonner Mathematische Schriften*. Bonn: Universität Bonn, 1987.

[24] M. Diehl, *Real-Time Optimization for Large Scale Nonlinear Processes*. PhD thesis, Universität Heidelberg, 2001. http://www.ub.uni-heidelberg.de/archiv/1659/.

[25] Sourceforge, "Acado toolkit." URL: `http://sourceforge.net/p/acado/wiki/Home/`, last checked on November 19, 2012.