High-Performance Small-Scale Solvers for Moving Horizon Estimation

Frison, Gianluca; Vukov, Milan; Poulsen, Niels Kjølstad; Diehl, Moritz; Jørgensen, John Bagterp

Published in:
Preprints of the 5th IFAC Conference on Nonlinear Model Predictive Control (NMPC)

Publication date:
2015

Document Version
Publisher’s PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
High-Performance Small-Scale Solvers for Moving Horizon Estimation

Gianluca Frison * Milan Vukov ** Niels Kjølstad Poulsen * Moritz Diehl ** *** John Bagterp Jørgensen *

* Technical University of Denmark (email: {giaf, nkpo, jbjo}@dtu.dk)
** KU Leuven (email: milan.vukov@esat.kuleuven.be)
*** University of Freiburg (email: moritz.diehl@imtek.uni-freiburg.de)

Abstract:
In this paper we present a moving horizon estimation (MHE) formulation suitable to easily describe the quadratic programs (QPs) arising in constrained and nonlinear MHE. We propose algorithms for factorization and solution of the underlying Karush-Kuhn-Tucker (KKT) system, as well as the efficient implementation techniques focusing on small-scale problems. The proposed MHE solver is implemented using custom linear algebra routines and is compared against implementations using BLAS libraries. Additionally, the MHE solver is interfaced to a code generation tool for nonlinear model predictive control (NMPC) and nonlinear MHE (NMHE). On an example problem with 33 states, 6 inputs and 15 estimation intervals execution times below 500 microseconds are reported for the QP underlying the NMHE.

© 2015, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

1. INTRODUCTION

Moving Horizon Estimation (MHE) has emerged as an effective option to state and parameter estimation of constrained or non-linear systems. It is found to give superior estimation performance with respect to the Extended Kalman Filter (EKF), at the cost of increased computational cost [11]; MHE requires the solution of an optimization problem at each sampling instant.

MHE can be seen as an extension of the Kalman Filter, where, beside the current measurement, a window of N past measurements is explicitly taken into account in the estimation. This makes the estimation less sensitive to the choice of the arrival cost, that rarely has an analytic expression in case of constrained or non-linear systems [18]. Furthermore, the MHE formulation can naturally and optimally take constraints into account.

From an algorithmic point of view, MHE is often considered the dual of Model Predictive Control (MPC), with the difference that the initial state is free. Therefore, algorithms for MPC have been used to solve MHE problems. In particular, a forward Riccati recursion (corresponding to a covariance Kalman filter recursion) has been proposed in [20; 14] for the solution of the unconstrained MHE subproblems. A QR factorization based, square-root forward Riccati is proposed as routine in an Interior-Point Method (IPM) for MHE in [12].

The focus of the current paper is on the computational performance of algorithms and implementations, rather than the control or estimation performance. More precisely, the focus is on the development of a fast solver for the equality-constrained linear MHE problem, specially tailored to small-scale problems. This solver is embedded in an algorithmic framework for non-linear MHE (presented in [15] and implemented using automatic code generation in [6]) and used to solve in real-time the QPs arising in equality-constrained non-linear MHE problems. The real-world test problem in Section 5.2 falls into this class of problems. Furthermore, the developed solver can be easily embedded as a routine into an IPM to solve inequality-constrained MHE problems, similarly to [9] for the MPC problem case. In an IPM, a solver for the equality-constrained MHE problem is used to compute the Newton direction, that is the most computationally expensive part of the algorithm. Hence the importance of a solver for this class of problems.

The focus on small-scale problems has important consequences on algorithmic and implementation choices. In case of small-scale dense problems (with dense meaning MPC and MHE problems where the dynamic system matrices are dense), solvers based on tailored recursions are much faster than general-purpose direct sparse solvers (see e.g. [7] for a comparison of a Riccati recursion based solver to PARDISO and MA57 direct sparse solves in the unconstrained MPC problem case). The performance gap suggests that direct sparse solvers may become competitive only for very sparse problems. In case of large-scale and sparse solvers, direct sparse solvers have been successfully applied to the MHE problem [23]. Furthermore, the focus on small-scale problems reduces the issues related to the numerical stability of the recursion schemes. It is well known that the Riccati recursion can be seen as a special stage-wise factorization of the KKT matrix of the unconstrained MPC problem. The factorization of different permutations of the KKT matrix can have better accuracy properties, especially in case of ill-conditioned problems.

In this paper, we study the applicability to the MHE problem of the efficient implementation techniques proposed in [9; 8] for the MPC problem, with special focus on small-scale performance. In particular, one of the key ingredients to obtain solvers giving high-performance for small matrices is the merging of linear algebra routines
is the covariance matrix of the process noise vector $w_k$. The vectors $\bar{v}_k$ and $\bar{w}_k$ are the expected values of the measurement and process noises.

In this paper, we consider a different formulation of the MHE problem. Namely, we consider a QP in the form

$$\min_{x_k,\bar{v}_k,\bar{w}_k} \sum_{k=0}^{N} \frac{1}{2} x_k^T Q_k x_k + \bar{v}_k^T x_k + \bar{w}_k^T \bar{w}_k + r_k^T w_k + \frac{1}{2} (x_0 - \bar{x}_0)^T P_0 (x_0 - \bar{x}_0)$$

s.t. $x_{k+1} = A_k x_k + G_k w_k + f_k$

$$y_k = C_k x_k + v_k$$

$$D_N x_N = d_N$$

The matrices $Q_k$, $R_k$ and $P_0$ can be interpreted as information matrices. In general, the matrices $R_k$ are assumed to be strictly positive definite, and the matrices $Q_k$ and $P_0$ are assumed to be positive semi-definite, with the matrix $Q_0 + P_0$ strictly positive definite.

This formulation reflects the deterministic view of the MHE as the problem of finding the optimal $x_k$, $w_k$ and $v_k$ sequences in a least-square sense, with respect to some cost function. The penalization of $x_k$ in place of $v_k$ in the cost function (1a) is useful to account for QPs in non-linear MHE. The fact that the matrices in the cost function appear as not-inverted makes straightforward the use of a solver for this MHE formulation as a routine for constrained MHE (e.g., in an IPM these matrices are updated to take into account constraints). The inversion does not need to be performed explicitly, but instead implicitly and embedded in the solution algorithm, as shown in section 4.1.

In this formulation, we consider additional state equality constraints (1d) beside the dynamic system equations (1b). These equality constraints are used to provide consistent feedback signal to the controller. They are enforced only at the last stage to avoid Linear Independence Constraint Qualification (LICQ) problems.

The size of problem (1) is defined by the quantities: $n_x$ (state vector size), $n_w$ (process noise vector size), $n_d$ (number of state equality constraints on the last stage), $N$ (horizon length).

3. STAGE-WISE FACTORIZATION OF THE KKT MATRIX

The MHE problem (1) is an equality constrained QP with a special structure. For $N = 2$, the solution is obtained solving the KKT (Karush-Kuhn-Tucker) system

$$\begin{bmatrix} E_0 & A_k^T & R_0 \end{bmatrix} \begin{bmatrix} x_0 & w_0 & \lambda_0 \end{bmatrix} = \begin{bmatrix} -e_0 & -t_0 & -d_0 \end{bmatrix},$$

where $\lambda_k$ are the Lagrangian multipliers and $E_0 = Q_0 + P_0$

$e_0 = q_0 + P_0 \bar{x}_0$

The KKT matrix is symmetric, large and structured. If the structure is not exploited, it can be factorized using a dense
The stage-wise structure of the KKT matrix can be exploited to factorize it stage-by-stage using a forward recursion, starting from the first stage. This recursion is analogue to the Information Filter (IF) formulation of the Kalman filter proposed in [16]. The recursion can be easily generalized (at the cost of a modest increase in the solution time) to handle a cross-term $S_k$ between $x_k$ and $u_k$ in the cost function. The top-left corner of the KKT matrix is

$$
\begin{bmatrix}
E_0 & A_0^T \\
R_0 & G_0^T \\
A_0 & G_0 \\
-I & I
\end{bmatrix}
\begin{bmatrix}
x_0 \\
w_0 \\
\lambda_0 \\
x_1
\end{bmatrix} =
\begin{bmatrix}
-e_0 \\
-w_0 \\
-r_0 \\
-f_0
\end{bmatrix}.
$$

(3)

If the matrix $E_0$ is invertible, the variable $x_0$ can be eliminated using the Schur complement of $E_0$, obtaining

$$
\begin{bmatrix}
R_0 & G_0^T \\
G_0 - A_0 E_0^{-1} A_0 & -I \\
-I & Q_1
\end{bmatrix}
\begin{bmatrix}
w_0 \\
\lambda_0 \\
x_1
\end{bmatrix} =
\begin{bmatrix}
-r_0 \\
-f_0 + A_0 E_0^{-1} e_0
\end{bmatrix}.
$$

Similarly, if the matrix $R_0$ is invertible, the variable $w_0$ can be eliminated, obtaining

$$
\begin{bmatrix}
-A_0 E_0^{-1} A_0^T - G_0 R_0^{-1} G_0 & -I \\
-I & Q_1
\end{bmatrix}
\begin{bmatrix}
\lambda_0 \\
x_1
\end{bmatrix} =
\begin{bmatrix}
-f_0 + A_0 E_0^{-1} e_0 + G_0 R_0^{-1} r_0
\end{bmatrix}.
$$

Finally, if the matrix $P_{k+1}^{-1} = A_0 E_0^{-1} A_0^T + G_0 R_0^{-1} G_0$ is invertible, the variable $\lambda_0$ can be eliminated, obtaining

$$(Q_1 + P_1) x_1 = -q_1 - P_1 (-f_0 + A_0 E_0^{-1} e_0 + G_0 R_0^{-1} r_0),$$

that can be rewritten in the more compact form

$$E_1 x_1 = -e_1$$

(4)

closing the recursion, since now the top-left corner of the KKT matrix is

$$
\begin{bmatrix}
E_1 & A_1^T \\
R_1 & G_1^T \\
A_1 & G_1 \\
-I & I
\end{bmatrix}
\begin{bmatrix}
x_1 \\
w_1 \\
\lambda_1 \\
x_2
\end{bmatrix} =
\begin{bmatrix}
-e_1 \\
-w_1 \\
-r_1 \\
-f_1
\end{bmatrix}.
$$

that is in the same form as (3). The recursion can therefore be repeated at the following stage. At the last stage, we can distinguish two cases, depending on the presence of equality constraints on the state vector at the last stage (1d).

If $n_d = 0$, the last stage looks like

$$E_2 x_2 = -e_2
$$

that, if $E_2$ is invertible, can be easily solved to compute $x_2$. Notice that the information matrix $E_2$ of the estimate $x_2$ is available at no extra cost.

If $n_d > 0$, the last stage looks like

$$
\begin{bmatrix}
E_2 & D_2^T \\
D_2 & D_2^T
\end{bmatrix}
\begin{bmatrix}
x_2 \\
\lambda_2
\end{bmatrix} =
\begin{bmatrix}
-e_2 \\
+d_2
\end{bmatrix}.
$$

If the matrix $E_2$ is invertible, the variable $x_2$ can be eliminated using the Schur complement of $E_2$, obtaining

$$-D_2 E_2^{-1} D_2^T \lambda_2 = d_2 + D_2 E_2^{-1} e_2.$$ If the matrix $D_2 E_2^{-1} D_2^T$ is invertible, then the value of $\lambda_2$ can be computed, that in turn gives the value of $x_2$ as

$$E_2 x_2 = -e_2 - D_2^T \lambda_2.$$ The information matrix of the estimate in the null-space can be computed as

$$E_{Z,Z} = Z^T E_2 Z$$

where $Z$ is a null-space matrix of $D$ [17].

Notice that the proposed recursion requires the invertibility of the matrices $R_k$ for $k = 0, \ldots, N - 1$, of the matrices $E_k = Q_k + P_k$ for $k = 0, \ldots, N$, of the matrices $P_{k}^{-1}$ (and then of the matrices $P_k$) for $k = 1, \ldots, N$, and of the matrix $D_N E_N^{-1} D_N^T$. However, the matrix $P_0$ can be singular: in particular, it can be set to 0 if no prior information is available about the value of the estimate of $x_0$. Invertibility of $Q_k$ for $k = 0, \ldots, N$ and full row-rank of $A_k$ for $k = 1, \ldots, N - 1$ and of $D_N$ guarantees the invertibility of $E_k$ for $k = 0, \ldots, N$, of $P_k$ for $k = 1, \ldots, N$ and of $D_N E_N^{-1} D_N^T$.

4. IMPLEMENTATION

In this paper, the efficient implementation techniques proposed in [9, 8] for the Riccati-based solver for the unconstrained MPC problem are applied to the MHE problem (1).

4.1 Algorithm

In the MPC case, the backward Riccati recursion can be seen as a stage-wise factorization of the KKT matrix, with the recursion beginning at the last stage [19]. The key operation in the algorithm presented in [9] is the computation of $Q + A^T \cdot P \cdot A$, where $Q$ is a positive semi-definite matrix. If all matrices $A$, $P$ and $Q$ have size $n$, then the most efficient way to compute this operation is

$$Q + A^T \cdot P \cdot A = Q + A^T \cdot (L \cdot L^T) \cdot A =$$

$$= Q + (A^T \cdot L) \cdot (A^T \cdot L)^T \hspace{1cm} (5)$$

where $L$ is the lower Cholesky factor of $P$. Using specialized BLAS routines, the cost of this operation is $\frac{3}{2} n^3$ (potrf) + $n^3$ (trsm) + $n^3$ (syrk) = $\frac{7}{2} n^3$ flops.

In the MHE case, in the forward recursion presented in Section 3 the key operation is the computation of $Q \cdot A^T \cdot P^{-1} \cdot A^T$, where $Q$ is a positive definite matrix. Despite the presence of a matrix inversion, this operation can be computed in the exact same number of flops as the operation in (5). In fact, the matrix inversion is computed implicitly, as

$$Q + A \cdot P^{-1} \cdot A^T = Q + A \cdot (L \cdot L^{-1}) \cdot A^T =$$

$$= Q + (A \cdot L^{-T}) \cdot (A \cdot L^{-T})^T \hspace{1cm} (6)$$

where again $L$ is the lower Cholesky factor of $P$. Since the matrix $L$ is triangular, the operation $A \cdot L^{-T}$ can be computed efficiently using the routine trsm to solve a triangular system of linear equations with matrix RHS. Using specialized BLAS routines, the cost of this operation is $\frac{3}{2} n^3$ (potrf) + $n^3$ (trsm) + $n^3$ (syrk) = $\frac{7}{2} n^3$ flops. This makes the IF-like recursion in Section 3 competitive with respect to the forward Riccati recursion generally used to factorize the KKT matrix of the MHE problem.

The algorithm for the factorization of the KKT matrix (2) is presented in Algorithm 1. The algorithm can be implemented using standard BLAS and LAPACK routines: the name of the routines is in the comment to each
Algorithm 1: Factorization of the KKT matrix of the MHE problem (1)

Require:
\[ \bar{U}_0 \text{ s.t. } P_0 = U_0 \cdot U_0^T \]

1: for \( k \leftarrow 0, \ldots, N - 1 \) do
2: \[ E_k \leftarrow \bar{Q}_k + U_k \cdot U_k^T \]
3: \[ L_{e,k} \leftarrow E_{e,k}^{1/2} \]
4: \[ A_{e,k} \leftarrow A_k \cdot L_{e,k}^T \]
5: \[ L_{r,k} \leftarrow R_k^{1/2} \]
6: \[ G_{r,k} \leftarrow G_k \cdot L_{r,k}^T \]
7: \[ P_{inv} \leftarrow A_{e,k} \cdot A_{L,e,k} + G_{r,k} \cdot G_{r,k}^T \]
8: \[ L_p \leftarrow P_{inv}^{1/2} \]
9: \[ U_{k+1} \leftarrow L_p \]
10: end for
11: \[ E_N \leftarrow Q_N - U_N \cdot U_N^T \]
12: \[ L_{e,N} \leftarrow E_{e,N}^{1/2} \]
13: if \( n_d > 0 \) then
14: \[ D_{L,e} \leftarrow D_N \cdot L_{e,N}^T \]
15: \[ P_d \leftarrow D_{L,e} \cdot D_{L,e}^T \]
16: \[ L_d \leftarrow P_d^{1/2} \]
17: end if

The cost of the algorithm is of \( O\left(\frac{n_0^2}{3} n_0^2 + n_0^2 n_w + n_0^2 n_w^2 + \frac{1}{3} n_0^3 + \frac{2}{3} n_0^3 + n_0 q n_0^2 + \frac{1}{3} n_0^3 + \frac{1}{3} n_0^3 \right) \) flops. If the \( R_k \) matrices are diagonal, then operations in lines 5 and 6 can be performed in a linear and quadratic number of flops, respectively. This decreases \( O\left(\frac{n_0^2}{3} n_0^2 + \frac{1}{3} n_0^3 + \frac{1}{3} n_0^3 \right) \) flops from the complexity of the algorithm, making it linear in \( n_0 \). This is advantageous in typical situations with MHE formulations involving additive process noise.

The algorithm for the solution of the KKT system given the factorization of the KKT matrix is presented in Algorithm 2. It consists of forward and backward substitutions. Again, triangular matrices are exploited by means of specialized routines.

4.2 Merging of linear algebra routines

All linear-algebra routines are implemented using the implementation techniques presented in [9; 8]. In particular, high-performance kernels for the general matrix-matrix multiplication routine \texttt{gemm} are used as the backbone of kernels for all matrix-matrix operations and factorizations. These kernels are optimized for a number of architectures, and can attain a large fraction of the floating-point (FP) peak performance. The design focus is on performance for small-scale matrices, but the performance scales optimally for matrices of size up to a few hundreds, large enough for embedded MPC and MHE needs.

In the optimization of solvers for small scale problems, it is beneficial to merge linear algebra routines when possible, as shown in the Riccati recursion for unconstrained MPC problems in [9]. The main advantage is the reduction in the number of calls to linear algebra kernels. In fact, in our implementation linear algebra kernels are blocked for register size, and therefore they compute a sub-matrix of the result matrix with a single kernel call. If the size of the result matrix is not a multiple of the optimal kernel size, there is a loss in performance: therefore merging small matrices into larger ones increases the likelihood of using the optimal kernel size. Furthermore, the reduction in the number of kernel calls reduces the corresponding overhead, and improves memory reuse. All these aspects are especially beneficial for small size problems.

As the problem size increases, however, the performance advantages of merging linear algebra routines become smaller, since the kernels call overhead gets amortized over a larger number of calls. On the contrary, numerical tests show that merging linear algebra routines often slightly decreases performance for large problems. This is due to the fact that merged routines operate on larger amounts of data than un-merged routines, and therefore cache size is exceeded for smaller problem sizes. The performance crossover point can be easily determined by numerical simulation, and it can be used as threshold to switch between merged and un-merged linear algebra routines.

In order to motivate the use of routine merging, let us consider a \( 3 \times 3 \) blocked version of the operation \( \mathcal{L} = (Q + A \cdot A^T)^{1/2} \) in (7). The last line contains the explicit expression of the lower Cholesky factor \( \mathcal{L} \): the expression for the \( L_{ij} \) block is in position \( ij \) in the matrix. We can see immediately that the products \( A_i \cdot A_j^T \) (used to compute the matrix to be factorized) are in the same form as the correction terms \( -L_{ik} \cdot L_{jk}^T \) in the Cholesky factorization (a part the change of sign). This means that the \( \mathcal{L} \) matrix can be computed sweeping it once block-wise: each block is initialized with \( Q_{ij} \), then updated with \( A_i \cdot A_j^T \) and corrected with the products \( -L_{ik} \cdot L_{jk}^T \), and finally Cholesky-factorized (diagonal blocks) or solved using a triangular matrix (off-diagonal blocks). So, diagonal blocks are computed using the merged kernel \texttt{syrk.potrf}, while the off-diagonal blocks are computed using the merged kernel \texttt{gemm_trsm}.
Having this in mind, lines 5, 6 of Algorithm 1 can be trivially merged: in fact, the `trsv` kernel is already used internally in the Cholesky factorization routine. This means that the operations in lines 5, 6 can be computed using a Cholesky-like factorization routine operating on rectangular matrices, as

\[
L = \begin{bmatrix} L_{00} & \ast & \ast \\ L_{20} & L_{11} & \ast \\ L_{22} & L_{21} & \ast \\
\end{bmatrix} = \left( \begin{bmatrix} Q_{00} & \ast & \ast \\ Q_{20} & Q_{31} & \ast \\ Q_{22} & Q_{21} & \ast \\
\end{bmatrix} + \begin{bmatrix} A_{0} \\ A_{1} \\ A_{2} \\
\end{bmatrix} \cdot \begin{bmatrix} A_{0}^{T} & A_{1}^{T} & A_{2}^{T} \\
\end{bmatrix} \right)^{1/2}
\]

\[
= \left( \begin{bmatrix} Q_{00} + A_{0} \cdot A_{0}^{T} & \ast & \ast \\ Q_{10} + A_{1} \cdot A_{0}^{T} + A_{1} \cdot A_{1}^{T} & Q_{11} + A_{1} \cdot A_{1}^{T} & \ast \\ Q_{20} + A_{2} \cdot A_{0}^{T} + A_{2} \cdot A_{1}^{T} + A_{2} \cdot A_{2}^{T} & Q_{21} + A_{2} \cdot A_{1}^{T} + A_{2} \cdot A_{2}^{T} & Q_{22} + A_{2} \cdot A_{2}^{T} \\
\end{bmatrix} \right)^{1/2}
\]

\[
= \left( \begin{bmatrix} Q_{00} + A_{0} \cdot A_{0}^{T} & \ast & \ast \\ Q_{10} + A_{1} \cdot A_{0}^{T} & Q_{11} & \ast \\ Q_{20} + A_{2} \cdot A_{0}^{T} & Q_{21} & Q_{22} + A_{2} \cdot A_{2}^{T} \\
\end{bmatrix} \right)^{1/2}
\]

5.1 Performance tests

The results of the tests reported in this section assess the performance of the proposed MHE solver when implemented using different libraries for linear algebra. Namely, the implementation using the custom and merged linear algebra routines presented in section 4.2 (that is part of the HPMPC toolbox [1]) is compared against two open source BLAS libraries: OpenBLAS and the Netlib BLAS.

OpenBLAS [3] is an highly optimized BLAS implementation, providing code tuned for a number of architectures. It is a fork of the successful (and now unsupported) GotoBLAS [10], and it supports also the most recent architectures. It makes use of a complex blocking strategy to optimize the use of caches and TLBs (Translation Lookaside Buffer), and key routines are written in assembly using architecture-specific instructions. Its performance is competitive against vendor BLAS. The version tested in this paper is the 0.2.14.

Netlib BLAS [2] is the reference BLAS. It is written in Fortran code and it is generic, not targeting any feature of specific architectures. It does not perform any blocking strategy, and level-3 routines are written as simple triple loops. The performance is usually poor for large matrices.

The test machine is a laptop equipped with the Intel Core i5 2410M processor, running at a maximum frequency of 2.9 GHz. The operating system is Linux Ubuntu 14.04, with gcc 4.8.2 compiler. The processor has 2 cores and 4 threads (however, only single-thread code is considered in our tests). The processor implements the Sandy Bridge architecture, supporting the AVX instruction set (that operates on 256-bit vector register, each holding 4 double or 8 single precision FP numbers). The Sandy Bridge core can perform one vector multiplication and one vector addition each clock cycle, and therefore in double precision it has a FP peak performance of 8 flops per cycle (that at 2.9 GHz gives 23.2 Gflops).

In Fig. 1 there is the result of a performance test. On the small scale (Fig. 1a), the performance of the HPMPC version is much better than both BLAS versions, and it can attain a large fraction of the FP peak performance for problems with tens of states. On the medium scale (Fig. 1b), the performance of HPMPC is steady at around 75-80% of FP peak, while the performance of the Netlib BLAS version is steady at around 15% of FP peak. On the other hand, the performance of OpenBLAS increases with the problem size. For even larger problems, the performance of unblocked implementations (HPMPC and Netlib BLAS) would decrease, while the performance of the OpenBLAS implementation would be steadily close to FP peak. Such large problem sizes are however of limited interest in embedded MHE, and therefore the HPMPC implementation gives the best performance for relevant problem sizes.
and taking into account the fact that this means that the operations in lines 2, 3, 4 can be upper triangular and the matrices, as that the operations in lines 5, 6 can be computed using a gemv trsv triangular.

Having this in mind, lines 5, 6 of Algorithm 1 can be where the product and is upper triangular. This is due to the fact that Algorithm 2 is implemented using level 2 BLAS, where matrices are streamed and there is no reuse in matrix elements. Therefore for large problems the substitution time is dominated by the cost of streaming matrices from main memory, that is the same for all implementations.

5.2 Nonlinear MHE and MPC in closed loop: real-time numerical simulations

In the following we present the strength of the presented solver for MHE for state estimation and control of a nonlinear system. Namely, we present results of closed-loop real-time simulations of rotational start-up for an airborne wind energy system [22]. The system is modeled as a differential-algebraic equation (DAE), with 27 differential equations of the form: 

\[ \frac{dx}{dt} = f(x, u, t) \]

where \( x \) is the state vector, \( u \) is the control input, and \( t \) is time. The system is subject to disturbances and external forces, which are modeled as \( v = g(x, u, t) \). The state estimation problem is formulated as a nonlinear least squares problem, which is solved using an iterative Newton-Raphson method. The control problem is formulated as a nonlinear optimization problem, which is solved using a sequential quadratic programming (SQP) method.

In essence, the execution times of the feedback estimate is obtained. Afterwards, the NMPC feedback step interval begins with a feedback step of the RTI scheme [4; 15]. The QP underlying the NMHE solver is solved using the implementation presented in Section 4, while the QP underlying the NMPC solver is handled with an efficient implementation from [9].

An augmented model used for the NMHE, one that includes a disturbance model, has \( n_x = 33 \) states and \( n_w = 6 \) disturbance inputs. Consistency conditions of the DAE model yield \( n_d = 9 \) equality constraints, while the number of estimation intervals is \( N = 15 \). On the other hand, the NMPC formulation has \( N = 50 \) intervals. For more details, we refer to [21] and references therein.

The simulation results are reported in Figure 3. A control interval begins with a feedback step of the RTI scheme for the NMHE (MHE FBK), after which the current state estimate is obtained. Afterwards, the NMPC feedback step is triggered (MPC FBK) for calculation of optimal control inputs. In essence, the execution times of the feedback steps amount to solutions of underlying QPs. After each feedback step corresponding preparation step is executed (MHE PREP and MPC PREP), which includes model integration, sensitivity generation and linearization of the objective and the constraints. In this setting both NMHE and NMPC run on the separate CPU cores.
The solution times for the feedback step of the NMHE are always less than 500 µs, and the maximum feedback times for the NMPC are always less than 3 ms. In total, the maximum feedback delay is always less than 3.5 ms, far below the control period of 40 ms. Note that in [21] qpOASES [5] solver is used to solve the QPs underlying the same NMHE formulation. In that case the feedback step of the NMHE alone requires about 3.5 ms, i.e. nearly seven times more than with the MHE QP solver proposed in this paper.

6. CONCLUSION

In this paper, we presented an information Kalman filter recursion for the MHE problem, that can be easily used as routine in constrained and non-linear MHE. Furthermore we proposed efficient implementation techniques tailored to this recursion form, with special focus on small-scale performance. The resulting solver is shown to give noticeable performance improvements when compared to the same algorithm implemented using optimized BLAS and LAPACK libraries. Furthermore, the solver has been used to solve QPs underlying a nonlinear MHE formulation and provides state estimates necessary for control of a challenging non-linear system in less than 500 µs.

REFERENCES