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

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-23 (2015) 080–086

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.

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

2405-8963
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2015.11.265

[8]. In the case of the MHE problem, the efficient Riccati recursion implementation proposed in [9] can not be employed, since the covariance of the state estimates (needed in the computation of the state estimates in the forward-backward substitutions) is never computed explicitly, but instead only implicitly in a Cholesky factorization. Therefore, in this paper we consider a different recursion for MHE, corresponding to the information Kalman filter. This recursion can be effectively implemented using the proposed techniques, giving noticeable speedups with respect to the use of optimized BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage) libraries, as shown in Section 5.1. As a further advantage, the use of this recursion as routine in solvers for constrained MHE is straightforward, since the inversion of the information matrices (updated at each iteration of e.g. an IPM) is efficiently embedded in the recursion itself. Finally, this recursion can naturally handle state equality constraints at the last stage, that are useful to provide the controller with consistent state feedback.

Besides the use of the efficient implementation techniques proposed in [9; 8], the key difference between the solver presented in the current paper and the one presented in [12] regards the choices at the linear algebra level. Given the generic matrices A and B, the upper Cholesky factor R of the matrix $A + B' \cdot B$ can be computed efficiently using the BLAS rank-k symmetric update routine `syrk` and LAPACK Cholesky factorization routine `potrf` at a cost of $\frac{4}{3}n^3$ flops (if all matrices are of size n). Given the uniqueness of the Cholesky factorization, R can also be computed using the LAPACK QR factorization routine `geqrf`, as $\begin{bmatrix} B \\ A^{1/2} \end{bmatrix} = Q \cdot R$, at the larger cost of $\frac{10}{3}n^3$ flops, plus possibly $\frac{1}{3}n^3$ to compute $A^{1/2}$. On the other hand, the QR factorization based on Householder reflections is more accurate, since the worse-conditioned normal matrix $B' \cdot B$ is not computed explicitly. The choice between the two implementations therefore depends on accuracy and speed requirements.

2. PROBLEM FORMULATION

The aim of the MHE problem is the reconstruction of the state vectors x_k, process noise vectors w_k and measurement noise vectors v_k, given the plant model, the measurement vectors y_k for a window of past time instants $k = 0, 1, \dots, N$ and an initial estimate of the state vector at time 0, $\bar{x}_0$, and relative covariance matrix $\widetilde{P}_0$, summarizing the contribution given by the measurements prior to time 0.

The (unconstrained) MHE problem is traditionally written as the Quadratic Program (QP)

$$\begin{aligned} \min_{x_k, w_k, v_k} \quad & \sum_{k=0}^{N} \tfrac{1}{2}(v_k - \bar{v}_k)^T \widetilde{R}_k^{-1}(v_k - \bar{v}_k) + \\ & + \sum_{k=0}^{N-1} \tfrac{1}{2}((w_k - \bar{w}_k)^T \widetilde{Q}_k^{-1}(w_k - \bar{w}_k) + \\ & + \tfrac{1}{2}(x_0 - \bar{x}_0)^T \widetilde{P}_0^{-1}(x_0 - \bar{x}_0) \\ s.t. \quad & x_{k+1} = A_k x_k + G_k w_k + f_k \\ & y_k = C_k x_k + v_k \end{aligned}$$

In this formulation, the inverse of the matrices in the cost function has a precise statistical interpretation: $\widetilde{R}_k$ is the covariance matrix of the measurement noise vector v_k, $\widetilde{Q}_k$ 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

$$\begin{aligned} \min_{x_k, w_k} \quad & \sum_{k=0}^{N} \tfrac{1}{2} x_k^T Q_k x_k + q_k^T x_k + \\ & + \sum_{k=0}^{N-1} \tfrac{1}{2} w_k^T R_k w_k + \\ & + r_k^T w_k + \tfrac{1}{2}(x_0 - \bar{x}_0)^T P_0 (x_0 - \bar{x}_0) && \text{(1a)} \\ s.t. \quad & x_{k+1} = A_k x_k + G_k w_k + f_k && \text{(1b)} \\ & y_k = C_k x_k + v_k && \text{(1c)} \\ & D_N x_N = d_N && \text{(1d)} \end{aligned}$$

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 lenght).

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

$$\left[\begin{array}{ccc|ccc|cc} E_0 & & A_0^T & & & & & \\ & R_0 & G_0^T & & & & & \\ A_0 & G_0 & & -I & & & & \\ \hline & & -I & Q_1 & & A_1^T & & \\ & & & & R_1 & G_1^T & & \\ & & & A_1 & G_1 & & -I & \\ \hline & & & & & -I & Q_2 & D_2^T \\ & & & & & & D_2 & \end{array}\right] \left[\begin{array}{c} x_0 \\ w_0 \\ \lambda_0 \\ \hline x_1 \\ w_1 \\ \lambda_1 \\ \hline x_2 \\ \lambda_2 \end{array}\right] = \left[\begin{array}{c} -e_0 \\ -r_0 \\ -f_0 \\ \hline -q_1 \\ -r_1 \\ -f_1 \\ \hline -q_2 \\ +d_2 \end{array}\right] \quad (2)$$

where λ_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

LDL factorization using $\frac{1}{3}((N-1)(2n_x+n_w)+n_x+n_d)^3$ flops. However, the problem structure can be exploited to greatly reduce this cost computational.

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 w_k in the cost function. The top-left corner of the KKT matrix is

$$\left[\begin{array}{ccc|c} E_0 & & A_0^T & \\ & R_0 & G_0^T & \\ A_0 & G_0 & & -I \\ \hline & & -I & Q_1 \end{array}\right] \left[\begin{array}{c} x_0 \\ w_0 \\ \lambda_0 \\ \hline x_1 \end{array}\right] = \left[\begin{array}{c} -e_0 \\ -r_0 \\ -f_0 \\ \hline -q_1 \end{array}\right]. \tag{3}$$

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

$$\left[\begin{array}{cc|c} R_0 & G_0^T & \\ G_0 & -A_0E_0^{-1}A_0 & -I \\ \hline & -I & Q_1 \end{array}\right] \left[\begin{array}{c} w_0 \\ \lambda_0 \\ \hline x_1 \end{array}\right] = \left[\begin{array}{c} -r_0 \\ -f_0 + A_0E_0^{-1}e_0 \\ \hline -q_1 \end{array}\right].$$

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

$$\left[\begin{array}{c|c} -A_0E_0^{-1}A_0^T - G_0R_0^{-1}G_0 & -I \\ \hline -I & Q_1 \end{array}\right] \left[\begin{array}{c} \lambda_0 \\ \hline x_1 \end{array}\right] = \\ = \left[\begin{array}{c} -f_0 + A_0E_0^{-1}e_0 + G_0R_0^{-1}r_0 \\ \hline -q_1 \end{array}\right].$$

Finally, if the matrix $P_1^{-1} = A_0E_0^{-1}A_0^T + G_0R_0^{-1}G_0$ is invertible, the variable λ_0 can be eliminated, obtaining

$$(Q_1 + P_1)\, x_1 = -q_1 - P_1(-f_0 + A_0E_0^{-1}e_0 + G_0R_0^{-1}r_0),$$

that can be rewritten in the more compact form

$$E_1 x_1 = -e_1 \tag{4}$$

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

$$\left[\begin{array}{ccc|c} E_1 & & A_1^T & \\ & R_1 & G_1^T & \\ A_1 & G_1 & & -I \\ \hline & & -I & Q_2 \end{array}\right] \left[\begin{array}{c} x_1 \\ w_1 \\ \lambda_1 \\ \hline x_2 \end{array}\right] = \left[\begin{array}{c} -e_1 \\ -r_1 \\ -f_1 \\ \hline -q_2 \end{array}\right].$$

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

$$\left[\begin{array}{cc} E_2 & D_2^T \\ D_2 & \end{array}\right] \left[\begin{array}{c} x_2 \\ \lambda_2 \end{array}\right] = \left[\begin{array}{c} -e_2 \\ +d_2 \end{array}\right].$$

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

$$(-D_2E_2^{-1}D_2^T)\lambda_2 = d_2 + D_2E_2^{-1}e_2.$$

If the matrix $D_2E_2^{-1}D_2^T$ is invertible, then the value of λ_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

$$E_{Z,2} = Z' E_2 Z$$

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 $\mathcal{Q} + \mathcal{A}^T \cdot \mathcal{P} \cdot \mathcal{A}$, where $\mathcal{Q}$ is a positive semi-definite matrix. If all matrices $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{Q}$ have size n, then the most efficient way to compute this operation is

$$\begin{aligned} \mathcal{Q} + \mathcal{A}^T \cdot \mathcal{P} \cdot \mathcal{A} &= \mathcal{Q} + \mathcal{A}^T \cdot (\mathcal{L} \cdot \mathcal{L}^T) \cdot \mathcal{A} = \\ &= \mathcal{Q} + (\mathcal{A}^T \cdot \mathcal{L}) \cdot (\mathcal{A}^T \cdot \mathcal{L})^T \end{aligned} \tag{5}$$

where $\mathcal{L}$ is the lower Cholesky factor of $\mathcal{P}$. Using specialized BLAS routines, the cost of this operation is $\frac{1}{3}n^3$ (`potrf`) + n^3 (`trmm`) + n^3 (`syrk`) = $\frac{7}{3}n^3$ flops.

In the MHE case, in the forward recursion presented in Section 3 the key operation is the computation of $\mathcal{Q} + \mathcal{A} \cdot \mathcal{P}^{-1} \cdot \mathcal{A}^T$, where $\mathcal{Q}$ 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

$$\begin{aligned} \mathcal{Q} + \mathcal{A} \cdot \mathcal{P}^{-1} \cdot \mathcal{A}^T &= \mathcal{Q} + \mathcal{A} \cdot (\mathcal{L} \cdot \mathcal{L}^T)^{-1} \cdot \mathcal{A}^T = \\ &= \mathcal{Q} + (\mathcal{A} \cdot \mathcal{L}^{-T}) \cdot (\mathcal{A} \cdot \mathcal{L}^{-T})^T \end{aligned} \tag{6}$$

where again $\mathcal{L}$ is the lower Cholesky factor of $\mathcal{P}$. Since the matrix $\mathcal{L}$ is triangular, the operation $\mathcal{A} \cdot \mathcal{L}^{-T}$ 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 $\frac{1}{3}n^3$ (`potrf`) + n^3 (`trsm`) + n^3 (`syrk`) = $\frac{7}{3}n^3$ 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:
$U_0 \quad s.t. \quad P_0 = U_0 \cdot U_0^T$

1: **for** $k \leftarrow 0, \dots, N-1$ **do**
2: $E_k \leftarrow Q_k + U_k \cdot U_k^T$ ▷ `lauum`
3: $L_{e,k} \leftarrow E_k^{1/2}$ ▷ `potrf`
4: $AL_{e,k} \leftarrow A_k \cdot L_{e,k}^{-T}$ ▷ `trsm`
5: $L_{r,k} \leftarrow R_k^{1/2}$ ▷ `potrf`
6: $GL_{r,k} \leftarrow G_k \cdot L_{r,k}^{-T}$ ▷ `trsm`
7: $P_{inv} \leftarrow AL_{e,k} \cdot AL_{e,k}^T + GL_{r,k} \cdot GL_{r,k}^T$ ▷ `syrk`
8: $L_p \leftarrow P_{inv}^{1/2}$ ▷ `potrf`
9: $U_{k+1} \leftarrow L_p^{-T}$ ▷ `trtri`
10: **end for**
11: $E_N \leftarrow Q_N + U_N \cdot U_N^T$ ▷ `lauum`
12: $L_{e,N} \leftarrow E_N^{1/2}$ ▷ `potrf`
13: **if** $n_d > 0$ **then**
14: $DL_e \leftarrow D_N \cdot L_{e,N}^{-T}$ ▷ `trsm`
15: $P_d \leftarrow DL_e \cdot DL_e^T$ ▷ `syrk`
16: $L_d \leftarrow P_d^{1/2}$ ▷ `potrf`
17: **end if**

line. The cost of the algorithm is of $N(\frac{10}{3}n_x^3 + n_x^2 n_w + n_x n_w^2 + \frac{1}{3}n_w^3) + \frac{2}{3}n_x^3 + n_d n_x^2 + n_d^2 n_x + \frac{1}{3}n_d^3$ 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 $N(n_x n_w^2 + \frac{1}{3}n_w^3)$ flops from the complexity of the algorithm, making it linear in n_w. 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 `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

Algorithm 2 Forward-backward substitution of the KKT system of the MHE problem (1)

Require:
$U_{k+1}, L_{e,k}, AL_{e,k}, L_{r,k}, GL_{r,k}, \quad k = 0, \dots, N-1$
$L_{e,N}, DL_e, L_d$

1: **for** $k \leftarrow 0, \dots, N-1$ **do**
2: $e_k \leftarrow q_k + U_k \cdot U_k^T \cdot \bar{x}_k$ ▷ `trmv`
3: $\bar{x}_{k+1} \leftarrow -f_0 + AL_{e,k} \cdot L_{e,k}^{-1} \cdot e_k$ ▷ `gemv & trsv`
4: $\bar{x}_{k+1} \leftarrow \bar{x}_{k+1} + GL_{r,k} \cdot L_{r,k}^{-1} \cdot r_k$ ▷ `gemv & trsv`
5: **end for**
6: $e_N \leftarrow q_N + U_N \cdot U_N^T \cdot \bar{x}_N$ ▷ `trmv`
7: **if** $n_d = 0$ **then**
8: $x_N \leftarrow -L_{e,N}^{-T} \cdot L_{e,N}^{-1} \cdot e_N$ ▷ `trsv`
9: **else**
10: $\lambda_N \leftarrow d_N + DL_e \cdot L_{e,N}^{-1} \cdot e_N$ ▷ `gemv & trsv`
11: $\lambda_N \leftarrow -L_d^{-T} \cdot L_d^{-1} \cdot \lambda_N$ ▷ `trsv`
12: $x_N \leftarrow -L_{e,N}^{-T} \cdot (e_N + DL_e^T \cdot \lambda_N)$ ▷ `gemv & trsv`
13: **end if**
14: **for** $k \leftarrow N-1, \dots, 0$ **do**
15: $\lambda_k \leftarrow U_k \cdot U_k^T \cdot (\bar{x}_{k+1} - x_{k+1})$ ▷ `trmv`
16: $x_k \leftarrow L_{e,k}^{-T} \cdot (-e_k - AL_{e,k}^T \cdot \lambda_k)$ ▷ `gemv & trsv`
17: $w_k \leftarrow L_{r,k}^{-T} \cdot (-r_k - GL_{r,k}^T \cdot \lambda_k)$ ▷ `gemv & trsv`
18: **end for**

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 flops. 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 3×3 blocked version of the operation $\mathcal{L} = (\mathcal{Q} + \mathcal{A} \cdot \mathcal{A}')^{1/2}$ in (7). The last line contains the explicit expression of the lower Cholesky factor $\mathcal{L}$: the expression for the $\mathcal{L}_{ij}$ block is in position ij in the matrix. We can see immmediately that the products $\mathcal{A}_i \cdot \mathcal{A}_j^T$ (used to compute the matrix to be factorized) are in the same form as the correction terms $-\mathcal{L}_{ik} \cdot \mathcal{L}_{jk}^T$ 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 $\mathcal{Q}_{ij}$, then updated with $\mathcal{A}_i \cdot \mathcal{A}_j^T$ and corrected with the products $-\mathcal{L}_{ik} \cdot \mathcal{L}_{jk}^T$, and finally Cholesky-factorized (diagonal blocks) or solved using a triangular matrix (off-diagonal blocks). So, diagonal blocks are computed using the merged kernel `syrk_potrf`, while the off-diagonal blocks are computed using the merged kernel `gemm_trsm`.

$$\begin{aligned}\mathcal{L} &= \begin{bmatrix}\mathcal{L}_{00} & * & * \\ \mathcal{L}_{10} & \mathcal{L}_{11} & * \\ \mathcal{L}_{20} & \mathcal{L}_{21} & \mathcal{L}_{22}\end{bmatrix} = \left(\begin{bmatrix}\mathcal{Q}_{00} & * & * \\ \mathcal{Q}_{10} & \mathcal{Q}_{11} & * \\ \mathcal{Q}_{20} & \mathcal{Q}_{21} & \mathcal{Q}_{22}\end{bmatrix} + \begin{bmatrix}\mathcal{A}_0 \\ \mathcal{A}_1 \\ \mathcal{A}_2\end{bmatrix} \cdot \begin{bmatrix}\mathcal{A}_0^T & \mathcal{A}_1^T & \mathcal{A}_2^T\end{bmatrix}\right)^{1/2} = \\ &= \left(\begin{bmatrix}\mathcal{Q}_{00} + \mathcal{A}_0 \cdot \mathcal{A}_0^T & * & * \\ \mathcal{Q}_{10} + \mathcal{A}_1 \cdot \mathcal{A}_0^T & \mathcal{Q}_{11} + \mathcal{A}_1 \cdot \mathcal{A}_1^T & * \\ \mathcal{Q}_{20} + \mathcal{A}_2 \cdot \mathcal{A}_0^T & \mathcal{Q}_{21} + \mathcal{A}_2 \cdot \mathcal{A}_1^T & \mathcal{Q}_{22} + \mathcal{A}_2 \cdot 2_0^T\end{bmatrix}\right)^{1/2} = \\ &= \begin{bmatrix}(\mathcal{Q}_{00} + \mathcal{A}_0 \cdot \mathcal{A}_0^T)^{1/2} & * & * \\ (\mathcal{Q}_{10} + \mathcal{A}_1 \cdot \mathcal{A}_0^T)\mathcal{L}_{00}^{-T} & (\mathcal{Q}_{11} + \mathcal{A}_1 \cdot \mathcal{A}_1^T - \mathcal{L}_{10} \cdot \mathcal{L}_{10}^T)^{1/2} & * \\ (\mathcal{Q}_{20} + \mathcal{A}_2 \cdot \mathcal{A}_0^T)\mathcal{L}_{00}^{-T} & (\mathcal{Q}_{21} + \mathcal{A}_2 \cdot \mathcal{A}_1^T - \mathcal{L}_{20} \cdot \mathcal{L}_{10}^T)\mathcal{L}_{11}^{-T} & (\mathcal{Q}_{22} + \mathcal{A}_2 \cdot \mathcal{A}_2^T - \mathcal{L}_{20} \cdot \mathcal{L}_{20}^T - \mathcal{L}_{21} \cdot \mathcal{L}_{21}^T)^{1/2}\end{bmatrix}\end{aligned} \tag{7}$$

Having this in mind, lines 5, 6 of Algorithm 1 can be trivially merged: in fact, the `trsm` 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 factorizatin routine operating on rectangular matrices, as

$$\begin{bmatrix}L_{r,k} \\ GL_r\end{bmatrix} = \text{rect\_potrf}\left(\begin{bmatrix}R_k \\ G_k\end{bmatrix}\right).$$

Lines 2, 3, 4 of Algorithm 1 perform a similar operation to the one in (7), with the difference that the $\mathcal{A}$ matrix is upper triangular and the $\mathcal{Q}$ and $\mathcal{L}$ matrices are rectangular. This means that the operations in lineas 2, 3, 4 can be computed as

$$\begin{bmatrix}L_{e,k} \\ AL_e\end{bmatrix} = \text{rect\_potrf}\left(\begin{bmatrix}Q_k \\ A_k\end{bmatrix} + \begin{bmatrix}U_k \\ 0\end{bmatrix} \cdot \begin{bmatrix}U_k^T & 0\end{bmatrix}\right),$$

where the product $U_k \cdot U_k^T$ takes into account the fact that U_k is upper-triangular.

Notice that, if a cross term S_k is present in the cost function, then operations in lines 2, 3, 4, 5, 6, plus the additional operations related to S_k can be merged in the single routine

$$\begin{bmatrix}L_{e,k} & * \\ L_{s,k} & L_{r,k} \\ AL_e & GL_r\end{bmatrix} = \text{rect\_potrf}\left(\begin{bmatrix}Q_k & * \\ S_k & R_k \\ A_k & G_k\end{bmatrix} + \begin{bmatrix}U_k \\ 0 \\ 0\end{bmatrix} \cdot \begin{bmatrix}U_k^T & 0\end{bmatrix}\right).$$

Lines 7, 8, 9 of Algorithm 1 can be merged as well. Lines 7, 8 implement the exact same operation in (7). The triangular matrix inversion and transposition in line 9 can be computed easily by considering the analogy of this operation with the `trsm` operation embedded in the Cholesky factorization. All operations in lines 7, 8, 9 can therefore be computed as

$$\begin{bmatrix}L_p \\ U_{k+1}\end{bmatrix} = \text{rect\_potrf}\left(\begin{bmatrix}0 \\ I\end{bmatrix} + \begin{bmatrix}AL_e & GL_r \\ 0 & 0\end{bmatrix} \cdot \begin{bmatrix}AL_e^T \\ GL_r^T\end{bmatrix}\right),$$

and taking into account the fact that U_{k+1} is upper triangular.

Similar arguments apply to the operations in the remaining lines 11, 12, 14, 15, 16 of Algorithm 1, and similarly the merged routine `gemv_trsv` can be used at lines 3, 4, 10, 12, 16, 17 of Algorithm 2.

5. NUMERICAL TESTS

5.1 Performance tests

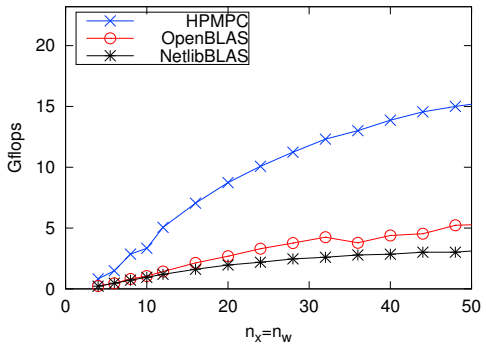
The results of the tests reported in this section assess the performance of the proposed MHE solver when implemented using different libraries for linear algebra. Namely, the implementation using the custom and merged linear algebra routines presented in section 4.2 (that is part of the HPMPC toolbox [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.

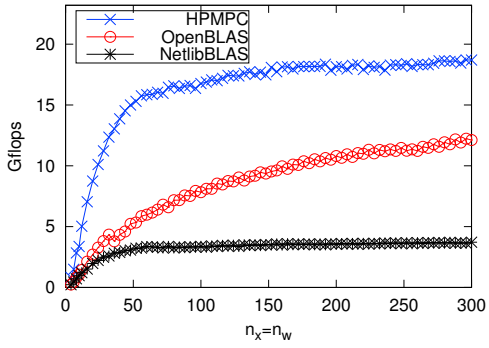
Netlib BLAS [2] is the reference BLAS. It is written in Fortran code and it is generic, not targeting any feature of specific architectures. It does not perform any blocking strategy, and level-3 routines are written as simple triple loops. The performance is usually poor for large matrices.

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.

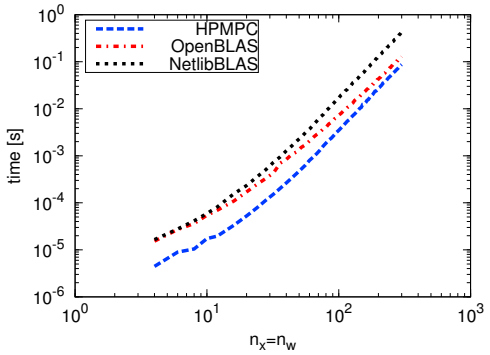


(a) Small scale.

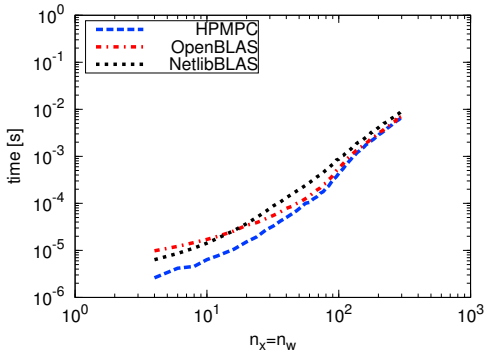


(b) Medium scale.

Fig. 1. Performance test for the proposed MHE KKT matrix factorization algorithm, assuming $S_k = 0$ and R_k dense. The performance in Gflops is represented as a function of $n_x = n_w$, while $N = 10$ and $n_d = 0$ are fixed. Top of the picture is the FP peak performance of the processor.



(a) Factorization time.



(b) Substitution time.

Fig. 2. Execution time for the proposed MHE KKT matrix factorization algorithm and forward-backward substitution algorithm, assuming $S_k = 0$ and R_k dense. The execution time in seconds is represented as a function of $n_x = n_w$, while $N = 10$ and $n_d = 0$ are fixed.

In Fig. 2 there are the running times for the factorization Algorithm 1 (Fig. 2a) and for the forward-backward substitution Algorithm 2 (Fig. 2b), in the three implementations using HPMPC, OpenBLAS and Netlib. In both the factorization and the substitution cases, the HPMPC implementation has a big advantage for small problems. In the factorization case, HPMPC retains the performance advantage over the Netlib BLAS version also for larger problems, while the the OpenBLAS version reduces the performance gap. In the substitution case, for larger problems the performance of the three implementations gets very similar. 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 states, 1 algebraic state and 4 control inputs. To solve the nonlinear MPC (NMPC) and nonlinear MHE (NMHE) formulations we use the ACADO Code Generation Tool (CGT) [13] that implements the real-time iteration (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.

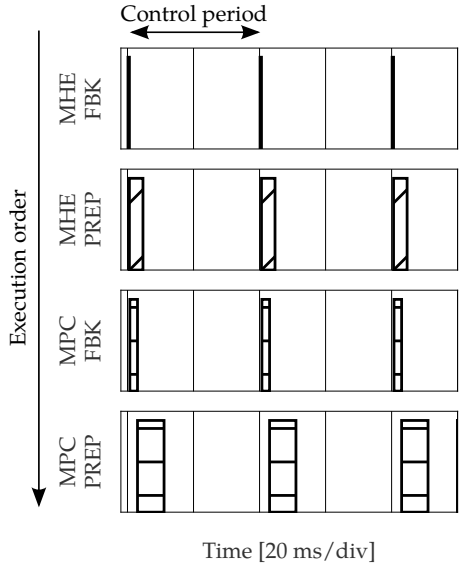


Fig. 3. Feedback and preparation step times for the MHE and MPC using the ACADO-HPMPC solver in the rotational start-up of an airbone wind energy system.

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

[1] HPMPC. https://github.com/giaf/hpmpc.git.
[2] Netlib BLAS. http://www.netlib.org/blas/.
[3] Openblas. http://www.openblas.net/.
[4] M. Diehl. *Real-Time Optimization for Large Scale Nonlinear Process*. PhD thesis, Universität Heidelberg, 2001.
[5] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control*, 18(8), 2008.
[6] H.J. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl. High-speed moving horison estimation based on automatic code generation. In *IEEE Conference on Decision and Control*, 2012.
[7] G. Frison and J. B. Jørgensen. Efficient implementation of the riccati recursion for solving linear-quadratic control problems. In *IEEE Multi-conference on Systems and Control*, pages 1117–1122. IEEE, 2013.
[8] G. Frison and J. B. Jørgensen. MPC related computational capabilities of ARMv7A processors. In *IEEE European Control Conference*. IEEE, 2015.
[9] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen. High-performance small-scale solvers for linear model predictive control. In *IEEE European Control Conference*, pages 128–133. IEEE, 2014.
[10] K. Goto and R. A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3), 2008.
[11] E. L. Haseltine and J. B. Rawlings. Critical evaluation of extended kalman filtering and moving-horizon estimation. *Ind. Eng. Chem. Res.*, 8(44):2451–2460, 2005.
[12] N. Haverbeke, M. Diehl, and B. De Moor. A structure exploiting interior-point method for moving horizon estimation. In *IEEE Conference on Decision and Control*, pages 1273–1278. IEEE, 2009.
[13] B. Houska, H. J. Ferreau, and M. Diehl. An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range. *Automatica*, 47(10):2279–2285, 2011.
[14] J. B. Jørgensen, J. B. Rawlings, and S. B. Jørgensen. Numerical methods for large-scale moving horizon estimation and control. In *Int. Symposium on Dynamics and Control Process Systems*, 2004.
[15] P. Kühl, M. Diehl, T. Kraus, J. P. Schlöder, and H. G. Bock. A real-time algorithm for moving horizon state and parameter estimation. *Computers & Chemical Engineering*, 1(35), 2011.
[16] G. O. Mutambara. *Decentralized estimation and control for multi-sensor systems*. CRC press, 1998.
[17] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.
[18] C. V. Rao, J. B. Rawlings, and Q. Mayne. Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations. *IEEE Transactions on Automatic Control*, 48(2), 2003.
[19] M. C. Steinbach. A Structured Interior-Point SQP Method for Nonlinear Optimal Control Problems. In *Computational Optimal Control*. Springer, 1994.
[20] J. Tenny, M. and J. B. Rawlings. Efficient moving horizon estimation and nonlinear model predictive control. In *American Control Conference*, 2002.
[21] M. Vukov. *Embedded Model Predictive Control and Moving Horizon Estimation for Mechatronics Applications*. PhD thesis, KU Leuven, April 2015.
[22] M. Zanon, S. Gros, and M. Diehl. Rotational Start-up of Tethered Airplanes Based on Nonlinear MPC and MHE. In *Proceedings of the European Control Conference*, 2013.
[23] V.M. Zavala and L.T. Biegler. Nonlinear programming strategies for state estimation and model predictive control. In *Nonlinear Model Predictive Control*, pages 419–432. Springer Berlin Heidelberg, 2009.