Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control

Alexander Domahidi[1], Aldo U. Zgraggen[1], Melanie N. Zeilinger[2], Manfred Morari[1] and Colin N. Jones[2]

***Abstract*— Receding horizon control requires the solution of an optimization problem at every sampling instant. We present efficient interior point methods tailored to convex multistage problems, a problem class which most relevant MPC problems with linear dynamics can be cast in, and specify important algorithmic details required for a high speed implementation with superior numerical stability. In particular, the presented approach allows for quadratic constraints, which is not supported by existing fast MPC solvers. A categorization of widely used MPC problem formulations into classes of different complexity is given, and we show how the computational burden of certain quadratic or linear constraints can be decreased by a low rank matrix forward substitution scheme. Implementation details are provided that are crucial to obtain high speed solvers. We present extensive numerical studies for the proposed methods and compare our solver to three well-known solver packages, outperforming the fastest of these by a factor 2-5 in speed and 3-70 in code size. Moreover, our solver is shown to be very efficient for large problem sizes and for quadratically constrained QPs, extending the set of systems amenable to advanced MPC formulations on low-cost embedded hardware.**

I. Introduction

Constrained finite time optimal control, also known as *model predictive control* (MPC) or *receding horizon control* (RHC), requires the solution of an optimization problem at each sampling instance. Efficient solvers with good numerical properties are therefore of paramount importance to enable reliable MPC implementations for fast systems on low-cost hardware. The computational burden of solving convex MPC problems online can be significantly reduced by exploiting the inherent problem structure [1]. Recent work showed that computational speeds in the millisecond range are possible [2]. The latest development is automatic code generation, where the computation times are further reduced by using a solver tailored to the specific optimization problem (CVXGEN, [3]). Although the results of CVXGEN are impressive for small-scale systems, severe limitations exist on the problem sizes that can be handled. The generated code can become of prohibitive size for larger problems (see Section V for example of what "large" means), limiting the applicability of the solver on low-cost microprocessors.

Furthermore, existing fast implementations of MPC controllers that are based on online optimization and run on embedded platforms support quadratic programs (QPs) only. As a result, important MPC formulations with desirable theoretical properties cannot be implemented in practice. Among these are formulations providing stability guarantees via ellipsoidal terminal sets [4], real-time stability guarantees [5] in face of early termination of the solver or robustness against modeling errors and noise in a tube MPC setting [6]. Moreover, some systems have inherent quadratic system constraints for which polytopic sets provide only a coarse approximation, such as power electronics [7].

This paper aims at closing the gap between MPC formulations with theoretically sound properties and current state of the art solver implementations, which are unable to solve these problems quickly and in a numerically stable manner on low-cost platforms. To that end, we discuss implementation strategies for efficient primal-dual interior point methods (IPMs), extending the set of MPC formulations that can be solved very efficiently. We consider the more general problem class of multistage optimization problems, which most relevant MPC formulations can be cast in. Among the variety of choices and subtleties in a particular implementation, we specify important algorithmic details that lead to a significant improvement in solver performance, both in terms of speed and numerical stability. In this context, we present a numerically stable and efficient rank one matrix forward substitution scheme, which is important to handle, for example, quadratic constraints efficiently. Although the result is an extension of [8], it has not been exploited in an MPC context to the authors' knowledge. Our implementation is shown to avoid the aforementioned limitations of existing solvers while outperforming them even for standard MPC problems such as QPs.

A variety of other methods and solvers have been proposed for solving linear MPC problems. For certain problem formulations and small dimensions, multi-parametric programming can be used to calculate the optimal state-to-input map offline, enabling model predictive control of extremely high-speed systems [9], [10]. In contrast to this so-called *explicit* MPC solution, online optimization can handle problems of any practically relevant dimension. First order methods are for certain problem classes the preferred choice, since tight bounds on the computation time can be provided [11], which, however, strongly depend on the conditioning of the problem. Primal-dual interior point methods show, albeit large (but polynomial) runtime bounds, generally a fast convergence in practice, independent of the problem data. In contrast to active set methods [12], which are essentially limited to linear and quadratic programs (LPs & QPs), IPMs can solve more general convex problems such as quadratically

[1]Automatic Control Laboratory, Department of Information Technology and Electrical Engineering, ETH Zurich, 8092 Zurich, Switzerland. domahidi|zgraggen|morari@control.ee.ethz.ch

[2]Automatic Control Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland. melanie.zeilinger|colin.jones@epfl.ch

constrained QPs (QCQPs) or second-order cone programs (SOCPs).

We present extensive numerical benchmarks of the well known interior point codes CPLEX [13] and CVXGEN [3] in comparison to our solver called FORCES, which has been written according to the guidelines presented in this paper. Furthermore, we include timings of the sparse factorization code MA57 [14] as called from OOQP [15] in order to show the merits of problem specific vs. general sparse factorization. While it is immediate that a customized solver, such as CVXGEN, is faster than general-purpose solvers, we show that FORCES is between 2-5 times faster than CVXGEN, which in turn is one to two orders of magnitude faster than CPLEX. As our results show, this speedup is achieved by structure exploitation and the fact that the FORCES code is of near-constant size even for large problem dimensions.

The paper is organized as follows: After introducing primal-dual IPMs (Section II), we discuss algorithmic details for the fast solution of multistage problems in Section III, including a cost analysis and a low rank modification scheme for matrix factorizations. Special cases relevant for common MPC formulations are pointed out. In Section IV, we discuss some important implementation aspects used in our code, which is shown to outperform the other solvers in Section V.

II. PRELIMINARIES

A. Problem formulation

In this paper, we consider the following general multistage optimization problem:

$$\begin{aligned} \min_{y_i, i=0\ldots N} \; & \sum_{i=0}^{N} l_i(y_i) \\ \text{s.t.} \quad & g_i(y_i) \leq 0 \;\; , \quad i = 0, \ldots, N \;\; , \\ & L_i\left(y_i, y_{i-1}\right) = 0 \;\; , \quad i = 1, \ldots, N \;\; , \end{aligned} \tag{P}$$

with $N+1$ *stage variables* $y_i \in \mathbb{R}^{p_i}$, convex stage cost functions $l_i : \mathbb{R}^{p_i} \to \mathbb{R}$ and $g_i : \mathbb{R}^{p_i} \to \mathbb{R}^{q_i}$ such that $\mathcal{Y}_i := \{y_i \mid g_i(y_i) \leq 0\}$ is a convex set with nonempty interior. The functions $L_i : \mathbb{R}^{p_i \times p_{i+1}} \to \mathbb{R}^{r_i}$ are affine, each depending on two consecutive stage variables:

$$L_i\left(y_i, y_{i-1}\right) := C_{i-1} y_{i-1} + D_i y_i + c_i \;\; ,$$

with $C_{i-1} \in \mathbb{R}^{r_i \times p_{i-1}}$, $D_i \in \mathbb{R}^{r_i \times p_i}$ such that $[C_{i-1} \; D_i]$ has full row rank and $c_i \in \mathbb{R}^{r_i}$. The optimization variable $y := \left[y_0^T, \ldots, y_N^T\right]^T$ is of dimension $p = \sum_{i=0}^{N} p_i$. We have $q = \sum_{i=0}^{N} q_i$ inequality constraints and $r = \sum_{i=0}^{N-1} r_i$ equality constraints in total.

We would like to point out that most relevant MPC problems can be cast into the form of (P), including problems with 1- or ∞-norm cost terms [16]. The necessary auxiliary variables become part of the stage variable, y_i. Problems with equality constrained initial state are captured as well as formulations where the initial state is an optimization variable, such as in robust MPC problems [6]. Moreover, (P) captures *moving horizon estimation* problems (MHE). The optimization problem (P) is convex (and potentially nonlinear) and can be solved by interior point methods.

B. Primal-dual interior point method

We solve (P) using a primal-dual interior point method employing Mehrotra's predictor-corrector scheme [17], which has proven to be very efficient in practice [18]. The particular algorithm follows mainly the lines of [1], but we perform the computation of the search direction by solving the KKT system in normal equations form as in [2] in order to fully exploit the specific problem structure. Details are given in the following.

a) KKT system: The KKT optimality conditions for (P) are given by

$$h(y) + C^T \nu + J(y)^T \lambda = 0 \;\; , \tag{1a}$$
$$Cy + c = 0 \;\; , \tag{1b}$$
$$g(y) + s = 0 \;\; , \tag{1c}$$
$$\Lambda S = 0 \;\; , \tag{1d}$$

where $s \in \mathbb{R}^q_{\geq 0}$ are slack variables, $\nu \in \mathbb{R}^r$ and $\lambda \in \mathbb{R}^q_{\geq 0}$ Lagrange multipliers and

$$h(y) := \left[\left(\nabla l_0(y_0)\right)^T, \ldots, \left(\nabla l_N(y_N)\right)^T\right]^T \in \mathbb{R}^p \;\; ,$$
$$C := \begin{bmatrix} C_0 & D_1 & 0 & \ldots & 0 \\ 0 & C_1 & D_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & C_{N-1} & D_N \end{bmatrix} \in \mathbb{R}^{r \times p} \;\; ,$$
$$J(y) := \text{blkdiag}\left(\nabla g_0(y_0), \; \ldots, \; \nabla g_N(y_N)\right) \in \mathbb{R}^{q \times p} \;\; ,$$
$$c := \left[c_0^T, \ldots, c_{N-1}^T\right]^T \in \mathbb{R}^r \;\; ,$$
$$g(y) := \left[g_0^T(y_0), \ldots, g_N^T(y_N)\right]^T \in \mathbb{R}^q \;\; ,$$
$$S := \text{diag}\left(s\right) \;\; , \qquad \Lambda := \text{diag}\left(\lambda\right) \;\; .$$

b) Search direction: Linearizing the KKT conditions gives rise to the Newton system

$$\begin{bmatrix} \mathcal{H}\left(y, \lambda\right) & C^T & J^T(y) & \\ C & & & \\ J(y) & & & I \\ & & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta \nu \\ \Delta \lambda \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_{\text{C}} \\ r_{\text{E}} \\ r_{\text{I}} \\ r_s \end{bmatrix} \;\; . \tag{2}$$

An IPM solves (2) for various right hand sides, which is the main computational burden of the algorithm. Efficient solution of (2) is therefore the key to obtain a high-performance solver. We show in Section III how to exploit the problem structure of (P) to solve (2) quickly and in a stable manner.

The $(1,1)$ block of the coefficient matrix in (2) is defined as

$$\mathcal{H}(y, \lambda) := \nabla h(y) + H(y, \lambda) \;\; , \tag{3a}$$
$$H(y, \lambda) := \text{blkdiag}\left(G_0(y, \lambda), \ldots, G_N(y, \lambda)\right) \tag{3b}$$
$$G_i(y, \lambda) := \sum_{j=1}^{q_i} \nabla^2 g_{i,j}(y_i) \cdot \lambda_{j+q_{i-1}} \in \mathbb{R}^{p_i \times p_i} \;\; , \tag{3c}$$

i.e. $G_i(y, \lambda)$ is the weighted sum of Hessians of the inequality constraints on stage variable i, weighted by the corresponding Lagrange multiplier.

c) Alternative KKT systems: Block elimination can be applied to (2) to obtain alternative formulations for the search direction computation. As the elements of Λ are strictly positive before convergence of the interior point method, we can eliminate Δs from (2) using $\Delta s = \Lambda^{-1}(r_s - S\Delta\lambda)$. This results in a *symmetric indefinite* system, which can be further reduced by eliminating the Lagrange multipliers:

$$\Delta\lambda = S^{-1}\Lambda\left(r_{\mathrm{I}} + J(y)\Delta y\right) + S^{-1}r_s \ . \tag{4}$$

We obtain the so-called *augmented system*

$$\begin{bmatrix}\Phi & C^T\\ C & 0\end{bmatrix}\begin{bmatrix}\Delta y\\ \Delta\nu\end{bmatrix} = -\begin{bmatrix}r_d\\ r_{\mathrm{E}}\end{bmatrix} \ , \tag{5}$$

where

$$\Phi := \mathcal{H}(y,\lambda) + J^T(y)S^{-1}\Lambda J(y) \ , \tag{6a}$$

$$r_d := h(y) + \mathcal{C}^T\nu + J^T(y)S^{-1}\Lambda r_{\mathrm{I}} \ . \tag{6b}$$

Finally, forming the Schur complement of the coefficient matrix in (5) gives the most compact form, often called *normal equations form* [18],

$$Y\Delta\nu = \beta \ , \tag{7a}$$

with

$$Y := C\Phi^{-1}C^T \in \mathbb{S}^r_+ \ , \tag{7b}$$

$$\beta := r_E - C\Phi^{-1}r_d \ , \tag{7c}$$

$$\Delta y = \Phi^{-1}\left(-r_d - C^T\Delta\nu\right) \ . \tag{7d}$$

We solve for the search direction in this form (7a) due to reasons outlined in Section III. For further details on the primal-dual IPM the reader is referred to [18].

III. Efficient solution of linearized KKT system

The most expensive step and performance bottleneck in any interior point method is the computation of the search direction (2). In this paper, we focus on solving the KKT system in normal equations form (7a) because of the following reasons. First, multi-stage problems of type (P) always yield a block banded coefficient matrix Y in (7a) that can be factored efficiently by block-wise Cholesky factorization [2]. This is due to the fact that the augmented Hessian Φ is block diagonal, hence its inverse again is block diagonal with blocks Φ_i^{-1}. Given the block-banded structure of C, Y therefore exhibits a symmetric block tri-diagonal structure. Second, Cholesky factorization is numerically stable without any pivoting, while a stable implementation of LDL^T factorization, which is needed to solve the augmented form (5), requires additional permutation strategies. Pivoting can produce significant overhead, since it is usually data-dependent and must therefore be carried out *on-the-fly*; fixed permutations as e.g. in CVXGEN are less favorable from a numerical point of view [19]. In our simulation studies, we did not experience any numerical problems due to ill conditioning when calculating (7b) or solving (7a), which might occur due to terms in $S^{-1}\Lambda$ that are close to zero at later iterations of the interior point method.

In the following, we outline the block Cholesky procedure from [2] to solve (7a) adapted to our more general problem (P) and give a cheaper method to compute Y.

A. Efficient calculation of Y

As stated above, Y has a block-banded structure:

$$Y := \begin{bmatrix} Y_{11} & Y_{12} & 0 & \dots & 0\\ Y_{12}^T & Y_{22} & Y_{23} & \dots & 0\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ 0 & 0 & Y_{N-2,N-1}^T & Y_{N-1,N-1} & Y_{N-1,N}\\ 0 & 0 & 0 & Y_{N-1,N}^T & Y_{N,N}\end{bmatrix} ,$$

with

$$Y_{i,i} := C_{i-1}\Phi_{i-1}^{-1}C_{i-1}^T + D_i\Phi_i^{-1}D_i^T \in \mathbb{R}^{r_i\times r_i} \ , \tag{8a}$$

$$Y_{i,i+1} := D_i\Phi_i^{-1}C_i^T \in \mathbb{R}^{r_i\times r_{i+1}} \ . \tag{8b}$$

The authors of [2] suggest to calculate (8) as follows. First, obtain the Cholesky factors L_i such that $\Phi_i = L_iL_i^T$. Then, a matrix forward- and backward-substitution is carried out to form $\tilde{C}_{i-1} := C_{i-1}\Phi_{i-1}^{-1}$ and $\tilde{D}_i := D_i\Phi_i^{-1}$, respectively. To obtain (8a), $\tilde{C}_{i-1}$ and $\tilde{D}_i$ are multiplied from the right by C_{i-1}^T and D_i^T, respectively; to obtain (8b), $\tilde{D}_i$ is multiplied by C_i^T.

However, we can do better than that as follows. After obtaining the Cholesky factor L_i as above, we solve

$$V_iL_i^T = C_i \ , \tag{9a}$$

$$W_iL_i^T = D_i \ , \tag{9b}$$

for $V_i \in \mathbb{R}^{r_i\times p_i}$ and $W_i \in \mathbb{R}^{r_i\times p_i}$ by matrix forward substitution. We now have a *rectangular factorization* of the quantities needed for (8), i.e.

$$C_{i-1}\Phi_{i-1}^{-1}C_{i-1}^T = V_{i-1}V_{i-1}^T \ , \tag{10a}$$

$$D_i\Phi_i^{-1}D_i^T = W_iW_i^T \ , \tag{10b}$$

$$D_i\Phi_i^{-1}C_i^T = W_iV_i^T \ . \tag{10c}$$

Based on elementary matrix operations in Table I, we compare the total cost of the two methods in Table II. The proposed method saves two matrix back-substitutions and ensures that $Y_{i,i}$ is symmetric even in case of rounding errors, which is not the case for [2] due to the asymmetric matrix products. Overall, our method saves $\left(p_{i-1}^2 + p_i^2\right)r_i$ floating point operations (flops[1]) and is numerically superior. Furthermore, it enables significant computational savings for special instances of (P) as described in Section III-C. These simplifications are possible only to a limited extent when using [2].

B. Block-wise Cholesky factorization of Y

After obtaining Y, we compute its Cholesky factor L_Y,

$$L_Y = \begin{bmatrix} L_{11} & 0 & 0 & \dots & 0\\ L_{21} & L_{22} & 0 & \dots & 0\\ 0 & L_{32} & L_{33} & \dots & 0\\ \vdots & \vdots & \ddots & \ddots & 0\\ 0 & 0 & \dots & L_{N,N-1} & L_N\end{bmatrix} ,$$

[1] addition, multiplication or division of two double precision numbers

TABLE I

COST OF ELEMENTARY MATRIX OPERATIONS WITH MATRICES $A, B : m \times n$, $C : n \times p$, $D, L : n \times n$, L LOWER TRIANGULAR.

Operation	Cost (flops)
Matrix matrix multiplication $A \cdot C$	$2mnp$
Symmetric matrix matrix multiplication $A \cdot A^T$	m^2n
LL^T decomposition s.t. $D = LL^T$	$2/3n^3$
Matrix forward substitution s.t. $AL^T = B$	mn^2
Matrix backward substitution s.t. $AL = B$	mn^2

TABLE II

COST FOR COMPUTING (8) WITH DIFFERENT METHODS (IN FLOPS).

Step	Operation	[2]	Operation	Proposed
1	Factor Φ_i $(= L_i L_i^T)$	$2/3p_i^3$	(same)	$2/3p_i^3$
2	$\tilde{C}_{i-1} := C_{i-1}\Phi_{i-1}^{-1}$	$2p_{i-1}^2 r_i$	Solve (9a)	$p_{i-1}^2 r_i$
3	$\tilde{D}_i := D_i \Phi_i^{-1}$	$2p_i^2 r_i$	Solve (9b)	$p_i^2 r_i$
4	$\tilde{C}_{i-1} C_{i-1}^T$	$p_{i-1} r_i^2$	(10a)	$p_{i-1} r_i^2$
5	$\tilde{D}_i D_i^T$	$p_i r_i^2$	(10b)	$p_i r_i^2$
6	$\tilde{D}_i C_i^T$	$2p_i r_i^2$	(10c)	$2p_i r_i^2$

by solving

$$Y_{11} = L_{11} L_{11}^T \ , \tag{11a}$$

$$Y_{i,i+1} = L_{i,i} L_{i+1,i}^T \ , \quad 1 \le i < N \ , \tag{11b}$$

$$Y_{i,i} - L_{i,i-1} L_{i,i-1}^T = L_{i,i} L_{i,i}^T \ , \quad 2 \le i \le N \ , \tag{11c}$$

where $L_{i,i} \in \mathbb{R}^{r_i \times r_i}$ are lower triangular and $L_{i+1,i} \in \mathbb{R}^{r_{i+1} \times r_i}$ are generally dense matrices. Equation (11b) is solved by matrix forward substitution, while (11a) and (11c) are solved by Cholesky factorizations of the corresponding matrices on the left hand side. This is a standard procedure and follows immediately from the structure of Y [2]. The total cost of this step is given in Table III.

We observe that the block-wise Cholesky factorization of Y amounts roughly to one third of the total costs, while the actual computation of the matrix to be factored, Y, is twice as expensive. Moreover, the problem data of (P) does not have a major influence on the cost of (11). This is not the case for (8), the cost of which is significantly dependent on the specific problem formulation. It is therefore worth investigating under which circumstances the computational burden can be lowered.

One special case is obviously that of Φ_i being diagonal, in which the cost of the first three steps in Table II is negligible. This holds also for the less obvious case when Φ_i is dense, but the sum of a diagonal and a rank one matrix, which will be shown in the following.

TABLE III

COSTS FOR THE BLOCK-WISE CHOLESKY FACTORIZATION OF Y.

Operation	Cost (flops)	
(11a)	$2/3r_1^3$	
(11b)	$r_i^2 r_{i+1}$	$1 \le i < N$
(11c)	$r_{i-1}^2 r_i + 2/3 r_i^3$	$2 \le i \le N$

C. Computational savings using rank one modifications

In this subsection, we describe how to efficiently compute (9) in certain cases. For generality, let us define a matrix

$$S := D + \alpha w w^T \in \mathbb{R}^{n \times n} \ , \tag{12}$$

with $D := diag(d_1, \ldots, d_n)$, $d_i, \alpha \in \mathbb{R}_{>0}$, and $w \in \mathbb{R}^n$. Let further $L \in \mathbb{R}^{n \times n}$ be the Cholesky factor of S, i.e. $S = LL^T$. The goal is to obtain $A \in \mathbb{R}^{m \times n}$ for some $B \in \mathbb{R}^{m \times n}$ such that

$$AL^T = B \ . \tag{13}$$

In this case, L can be computed at a cost of order n^2 by rank one modification methods [8]. While this cost is cheaper than a direct factorization of S, obtaining A still requires a matrix forward substitution at the cost mn^2.

It is easy to show that with the formula by *Sherman and Morrison* [20], calculating A directly (without calculating L first) is possible at an overall cost of order mn. However, it is well known that [20] is an unstable method [21], which we also observed in our experiments. We provide an efficient and stable method in the following that is based on [8] and makes use of the special structure of L:

$$L_{i,i} = \sqrt{d_i + \alpha_i w_i^2} \ , \qquad 1 \le i \le n \tag{14a}$$

$$L_{i,j} = \beta_j w_i \ , \qquad j < i \le n \ , \tag{14b}$$

where

$$\beta_j := \alpha_j w_j / L_{j,j} \ , \qquad 1 \le j \le n \ , \tag{14c}$$

$$\alpha_{j+1} := d_j \alpha_j / L_{j,j}^2 \ , \qquad 1 \le j < n \ , \tag{14d}$$

and $\alpha_1 = \alpha$. In words, the sub-diagonal elements of column j of L are a multiple of the corresponding elements of w, see (14b). We can exploit this property to perform an efficient matrix forward substitution while computing L at the same time. The standard matrix forward substitution formula for the elements of A is given by

$$a_{i,j} = \frac{b_{i,j} - \sum_{k=1}^{j-1} a_{i,k} L_{j,k}}{L_{j,j}} \ . \tag{15}$$

Substituting (14), we obtain

$$a_{i,j} = \frac{b_{i,j} - w_j \sum_{k=1}^{j-1} a_{i,k} \beta_k}{\sqrt{d_j + \alpha_j w_j^2}} \ , \tag{16}$$

where w_j can been moved in front of the sum by property (14b). Introducing the variable

$$\gamma_{i,j} := \sum_{k=1}^{j-1} a_{i,k} \beta_k \ , \tag{17}$$

a recurrence relation can be established between $\gamma_{i,j}$ and $\gamma_{i,j+1}$:

$$\gamma_{i,j+1} = \gamma_{i,j} + a_{i,j} \beta_j \tag{18}$$

with $\gamma_{i,1} = 0$. This allows to avoid the innermost loop of the matrix forward substitution, i.e. the summation in (15),

if A is calculated row-wise with increasing column index j. Hence we can obtain A at a total cost of order mn (instead of $n^2 + mn^2$) in a numerically stable fashion.

Concerning the computational complexity of (8) in Table II, the above implies that whenever either Φ_{i-1} or Φ_i have the structure of (12), the costs for obtaining (9a) and (9b) are negligible. Moreover, also the cost of the first step, i.e. for obtaining the Cholesky factor L_i, can then be neglected. These are generally significant savings which can be obtained without numerical degradation.

D. Problem dependent complexity categories

As we have seen in Section III-A and Section III-B, the overall cost of the interior point method depends on the cost of computing Φ_i, Y_i, and $Y_{i,i+1}$, which depend on the problem formulation, and on factoring Y, which is independent of latter. Starting from (6a), we note that there are the following five basic complexity levels (the analysis is carried out block-wise, denoted by subscripts i):

(A) $\mathcal{H}_i(y_i, \lambda_i)$ diagonal and $J_i^T(y_i) S_i^{-1} \Lambda_i J_i(y_i)$ diagonal $\forall y_i$ results in Φ_i diagonal. Operations related to factoring Φ_i as well as matrix forward and back substitutions to obtain products by Φ_i^{-1} become negligible (Steps 1-3 in Table II).

(B) $\mathcal{H}_i(y_i, \lambda_i)$ diagonal and $J_i^T(y_i) S_i^{-1} \Lambda_i J_i(y_i)$ low rank $\forall y_i$ results in dense Φ_i, but inexpensive low rank modifications as shown in Section III-C can be used such that Steps 1-3 in Table II can be neglected. (This case is however more expensive than case A due to squared cost terms).

(C) $\mathcal{H}_i(y_i, \lambda_i)$ dense but such that $\mathcal{H}_i(y_i, \lambda_i) = H_i + \sum_{j=1}^{q_i} \lambda_i H_i$, and $J_i^T(y_i) S_i^{-1} \Lambda_i J_i(y_i)$ low rank. Matrix H_i can be pre-factored offline, $H_i = L_{H_i} L_{H_i}^T$, and stored. The Cholesky factor of Φ_i can then be computed by a low rank modification of $L_{H_i}(\lambda_i) := L_{H_i} \left(I + \sum_{j=1}^{q_i} \lambda_i I \right)^{-1/2}$ at a cost of order p_i^2. This procedure removes Step 1 in Table II.

(D) $\mathcal{H}_i(y_i, \lambda_i)$ dense and $J_i^T(y_i) S_i^{-1} \Lambda_i J_i(y_i)$ low rank. All steps of Table II are necessary, but computing Φ_i is of order p_i^2.

(E) All other cases: All steps of Table II are necessary, and computing Φ_i additionally adds a cost of $p_i^2 r_i$ due to the term $J_i^T(y_i) S_i^{-1} \Lambda_i J_i(y_i)$.

E. Application to Linear MPC: Special Cases

In this section, we make the connection from Section III-D to MPC and show where the previously discussed cases are relevant when solving an MPC problem. Table IV categorizes widely used MPC problems into the problem classes introduced in Section III-D. Interestingly, the most expensive case is the one with dense linear constraints, as often used in MPC formulations with polytopic sets. The least expensive problem occurs for box constraints and diagonal costs, while problems with quadratic constraints are in between these two extremes. We would like to point out that the common case of a quadratic terminal cost, $l_N(x_N) := x_N^T P x_N$, along with a level set of latter as terminal set constraint, $x_N^T P x_N \leq \alpha$, falls into category C. This is an important result, since computation of polytopic terminal sets (category E) are prohibitively complex for high dimensions and quadratic terminal sets therefore represent a computationally beneficial alternative that can be applied to all problem dimensions.

TABLE IV
CATEGORIZATION OF MPC PROBLEMS INTO COMPLEXITY CLASSES.

$g_i(y_i) \setminus l_i(y_i)$	$c^T y_i$	$y_i^T Q y_i$, Q diag.	$y_i^T Q y_i$, Q dense
$\underline{y}_{ij} \leq y_{ij} \leq \bar{y}_{ij}$	A	B	E
$F_i y_i \leq f_i$, F dense	E	E	E
$y_i^T M_{ij} y_i \leq r_{ij}$	B	B	D (C if $Q = M_{i,j}$)

Note that the structure of the equality constraints in an MPC problem allows for a further complexity reduction. The associated matrices can be defined for standard dynamics $x_{k+1} = Ax_k + Bu_k$ as

$$C_0 := [I \ 0_{n_x \times n_u}], \qquad D_1 := 0_{n_x \times (n_x + n_u)} \ ,$$
$$C_i := [A \ B], \qquad D_{i+1} := [-I_{n_x} \ 0_{n_x \times n_u}], i = 2...N.$$

If the states and inputs are separable, Φ_i has block structure with block sizes n_x and n_u, respectively. In this case each operation in Table II decomposes into two cheaper operations, one with $p_i = n_x$, $r_i = n_x$ and one of size $p_i = n_u$, $r_i = n_x$. Furthermore, in the case of diagonal costs and box constraints, all matrices involved in (10b) and (10c) are diagonal, hence the cost of Step 5 and 6 of Table II can be neglected.

To summarize, we have extended the tailored Newton step computation introduced in [2] to the more generic problem class (P) and provided a detailed cost analysis of the search direction computation. Based on this analysis, special cases of (P) which allow for significant computational savings were identified. Among these are problems with box constraints, diagonal costs and, most importantly, quadratic constraints which are preferable in MPC formulations with stability guarantees. These can be in fact cheaper than formulations with general polytopic constraints, which, despite their wide usage, belong to the most expensive category.

IV. IMPLEMENTATION DETAILS: FAST AND SMALL CODE

The results presented in the next section have been obtained with a code written in strict ANSI-C in order to support a wide range of embedded platforms. For high performance, the operations for computing Φ, Y, L_Y and the forward substitutions needed for $\Delta\nu$ are carried out block-wise; for instance, instead of computing Y as a whole matrix, we directly factor Y_{ii} as soon as it is available and perform the forward substitution. This interleaving, or "zipping" of basic linear algebra operations greatly enhances both temporal and spatial locality of the code, reducing cache misses to a minimum. Note that the matrices involved in the computation of (8) and (11) are of small dimension (in case of linear MPC with quadratic costs: n_x and n_u) and thus typically fit into the first level cache. Therefore, slow data transfers to or from main memory are significantly reduced and the CPU or DSP can be kept as busy as possible.

Our solver builds on top of a small linear algebra library for operations on symmetric matrices, which we store in lower triangular format. We use static memory allocation, and store the working set of the code in simple C arrays. Computations were cached in memory if the result is needed at different places of the solver (e.g. the term $S^{-1}\Lambda$).

Despite the aforementioned measures for high performance, we would like to point out that we use standard, naive code with nested `for`-loops for our linear algebra system. Since the matrix dimensions are fixed a-priori, it is easy for the compiler to perform loop unrolling optimizations. As a result, our code is very small and library free[2]. A direct comparison to `MA57BD`, a widely used LDL^T sparse factorization code, is given in the next section.

V. Computational Results

A. The oscillating masses benchmark problem

Benchmark problem 1 (BP1). In order to evaluate the performance of the solver for various problem sizes, the following MPC problem is formulated for a chain of masses interconnected with spring-dampers [2]:

$$\begin{aligned} \min_{\mathbf{u}} \; & V_N(x,\mathbf{u}) := \sum_{n=0}^{N-1} x_n^T Q x_n + u_n^T R u_n + V_f(x_N) \\ \text{s.t. } & x_0 = x(0) \;, \\ & x_{n+1} = A_n x_n + B_n u_n \qquad n < N \;, \\ & -4\cdot\mathbf{1}_{n_x} \le x_n \le 4\cdot\mathbf{1}_{n_x} \qquad n \le N \;, \\ & -0.5\cdot\mathbf{1}_{n_u} \le u_n \le 0.5\cdot\mathbf{1}_{n_u} \quad n < N \;, \end{aligned} \tag{19}$$

with $R = I$ and $Q = 3I$ and $V_f(x_N) := x_N^T Q x_N$. Note that this formulation does not provide stability guarantees since no terminal constraint is included. The problem size can be chosen by means of the number of masses M, with the relation $n_x = 2M$ and $n_u = M - 1$. The dynamic matrices are dense due to discretization (time step 0.5s). The problem was solved for various numbers of masses M and prediction horizons N on 3 different platforms for a test set of 100 randomly chosen feasible points. We formulate the problem as a sparse QP, leaving the states as optimization variables. This benchmark problem belongs to class A (see Section III-D) and has a total cost of $(N+1)(8/3n_x^3 + n_x^2 n_u) + N n_x^3$ flops per interior point iteration.

Benchmark problem 2 (BP2). To show the code performance for a more complex problem, we add a quadratic terminal cost $V_f(x_N) := x_N^T P x_N$ and a quadratic terminal constraint $x_N^T P x_N \le \alpha$ to (19), where P solves the discrete-time Riccati equation related to LQR control and α determines the maximum level set of V_f such that no constraints are violated. In addition, we add a real-time constraint $V_N(x) \le \tau$ to ensure that the system is stable even in case of early termination of the solver, see [5] for details. The parameter τ is the cost of the previous solution minus a small multiple of the stage cost $x_0^T Q x_0$. To exploit the block structure in (P), we express $V_N(x) \le \tau$ by

$$x_n^T Q x_n \le \gamma_n \;, \quad u_n^T R u_n \le \delta_n \;, \quad n < N \;, \tag{20a}$$

$$J_{n+1} = J_n - \gamma_n - \delta_n \;, \qquad n < N \;, \tag{20b}$$

$$J_0 = \tau \;, \quad x_N^T P x_N - J_N \le 0 \;. \tag{20c}$$

Note that this results in a QCQP with block sizes n_x+1 and n_u+1. The rank-1 modification scheme of Section III-C can be used for the quadratic constraints. The problem belongs to complexity class B for the first N blocks, the last block is class C.

[2]A *square root* function is needed, which is available on all platforms

TABLE V
Code size of compiled code on desktop PC.

MPC Problem				Code Size [kB]		
M	n_x	n_u	N	MA57 library	CVXGEN	FORCES
2	4	1	10	1102	318	104
4	8	3	10	1102	848	134
6	12	5	10	1102	1885	156
11	22	10	10	1102	7654	127
15	30	14	10	1102	#	131
30	60	29	30	1102	#	191

B. Solvers and compilers

In the following, we compare the solve times of the FORCES code against those of CPLEX and CVXGEN. To assess the performance of the block-wise Cholesky factorization with respect to standard sparse LDL^T factorization, we measure the time needed for latter using the MA57 library, which is currently one of the best sparse factorization codes.

On the desktop PC and the Atom platform, we have used the *Intel C Compiler 12.0.3* with options `-O3` and, depending on the CPU architecture `-m64` for 64 bit or `-m32` for 32 bit, to compile all solvers except CPLEX 12.2, which comes in binary format. *Sourcery G++ Lite 4.5.2* was used for cross compilation for the ARM platform with flag `-O3`. We compiled MA57 (version 3.7.0) using the *Intel Fortran Compiler 12.1.0* and the *Intel Math Kernel Library* that is shipped with *Intel Composer XE 2011 SP1 7.256*. OOQP 0.99.22 was used to call MA57.

Remark V.1 *Note that some software packages, such as* OOQP*, can be supplied with a custom linear algebra system. In these cases, the methods proposed in this paper could be implemented in order to speed up computations.*

C. Code performance

The results of our numerical case study are summarized in Table VI for BP1. The last column gives the computation times for BP2 when solved by our solver FORCES (denoted by RT for real-time formulation). BP2 cannot be solved by OOQP and CVXGEN due to the quadratic constraints; the timings for BP2 solved by CPLEX have been omitted.

For small problem sizes, the tailored solvers CVXGEN and FORCES outperform CPLEX by two orders, and MA57 by one order of magnitude in speed, although we compare the factorization time vs. *full* interior point iterations. As the problem size increases, CPLEX scales very well, while code

TABLE VI

RUN TIMES FOR 10 INTERIOR POINT ITERATIONS (FOR MA57: 10 FACTORIZATIONS), AVERAGED OVER 100 RANDOM INITIAL STATES.

	MPC Problem				Optimization Problem			Runtimes [ms]				
Platform	M	n_x	n_u	N	p	r	q	CPLEX	MA57	CVXGEN	FORCES	FORCES RT
	2	4	1	10	54	44	108	10.73	1.60	0.18	0.09	0.25
Desktop PC	4	8	3	10	118	88	236	10.80	5.11	0.71	0.27	0.67
Intel Core i7	6	12	5	30	522	372	1044	16.20	31.45	5.90	1.95	4.15
3.2 GHz, 12 GB RAM	11	22	10	10	342	242	684	16.60	29.19	17.28	2.98	5.53
Ubuntu Linux 10.04	15	30	14	10	470	330	940	23.20	59.87	#	6.37	11.63
	30	60	29	30	2730	1860	5460	177.70	686.12	#	126.43	218.94
	2	4	1	10	54	44	108	*	*	1.62	0.77	2.06
Embedded 1	4	8	3	10	118	88	236	*	*	5.77	2.33	5.77
Intel Atom Z530	6	12	5	30	522	372	1044	*	*	48.07	15.71	35.46
1.6 GHz, 2GB RAM	11	22	10	10	342	242	684	*	*	†	22.83	43.03
Ubuntu Linux 11.04	15	30	14	10	470	330	940	*	*	#	48.97	90.71
	30	60	29	30	2730	1860	5460	*	*	#	984.96	1676.82
	2	4	1	10	54	44	108	*	*	62.09	24.61	59.73
Embedded 2	4	8	3	10	118	88	236	*	*	245.65	93.58	174.68
ARM Cortex A8 (TI OMAP 3530)	6	12	5	30	522	372	1044	*	*	†	662.23	1157.98
500 MHz, 256 MB RAM	11	22	10	10	342	242	684	*	*	†	933.38	1461.68
Ångström Linux 2.6.32	15	30	14	10	470	330	940	*	*	#	2072.76	3208.53
	30	60	29	30	2730	1860	5460	*	*	#	39505.55	61348.73

* No running implementation † Unable to compile # Unable to generate code

generation with CVXGEN fails for the last three problems. It can be seen that FORCES is the fastest solver out of the 4 candidates, solving the problem at least twice as quickly as CVXGEN. This gap grows with the size of the problem, as CVXGEN generates more and more code which has to be loaded into the instruction decoding unit of the CPU. A comparison of code sizes of the binary code is given in Table V, showing that FORCES is the smallest code. Consequently, for the 11-masses problem, the proposed solver is more than 5 times faster than CVXGEN, and it scales extremely well with the problem size. Furthermore, the problem formulation with superior theoretical properties (BP2) is solved more quickly by FORCES than BP1, which does not provide stability guarantees, by CVXGEN.

ACKNOWLEDGEMENT

We would like to thank Paul J. Goulart for helpful discussions during the course of this work and Markus Püschel for teaching us how to write fast numerical code.

The research leading to these results has received funding from the EU in FP7 via EMBOCON (ICT-248940).

REFERENCES

[1] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, pp. 723–757, 1998.

[2] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267 –278, March 2010.

[3] J. Mattingley, Y. Wang, and S. Boyd, "Receding horizon control: Automatic generation of high-speed solvers," in *IEEE Control Systems Magazine*, vol. 31, no. 3, June 2011, pp. 52–65.

[4] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789 – 814, 2000.

[5] M. N. Zeilinger, C. N. Jones, D. Raimondo, and M. Morari, "Real-time MPC - stability through robust MPC design," in *Proc. of the 48th IEEE Conf. on Decision and Control*, dec. 2009, pp. 3980 –3986.

[6] D. Mayne, M. Seron, and S. Rakovi, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219 – 224, 2005.

[7] G. Beccuti, G. Papafotiou, and L. Harnefors, "Multivariable predictive control of voltage source converter HVDC transmission systems," in *IEEE International Symposium on Industrial Electronics (ISIE)*, July 2010, pp. 3145 –3152.

[8] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, "Methods for modifying matrix factorizations," *Mathematics of Computations*, vol. 28, pp. 505–535, 1974.

[9] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, January 2002.

[10] S. Mariéthoz, A. Domahidi, and M. Morari, "Sensorless explicit model predictive control of permanent magnet synchronous motors," in *IEEE International Electric Machines and Drives Conference (IEMDC)*, May 2009, pp. 1250 –1257.

[11] S. Richter, C. N. Jones, and M. Morari, "Real-time input-constrained MPC using fast gradient methods," in *Proc. of the 48th IEEE Conference on Decision and Control*, Dec. 2009, pp. 7387 –7393.

[12] H. Ferreau, H. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control*, vol. 18, no. 8, pp. 816–830, 2008.

[13] [Online]. Available: http://www.ibm.com/software/integration/optimization/cplex-optimization-studio/

[14] I. S. Duff, "MA57 – a code for the solution of sparse symmetric definite and indefinite systems," *ACM Trans. Math. Softw.*, vol. 30, pp. 118–144, June 2004.

[15] E. M. Gertz and S. J. Wright, "Object-oriented software for quadratic programming," *ACM Transactions on Mathematical Software*, vol. 29, pp. 58–81, March 2003.

[16] F. Borrelli, *Constrained optimal control of hybrid systems*. Springer, 2003, vol. 290.

[17] S. Mehrotra, "On the implementation of a primal-dual interior point method," *Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[18] S. J. Wright, *Primal-dual interior point methods*. SIAM, 1997.

[19] J. W. Demmel, *Applied numerical linear algebra*. SIAM, 1997.

[20] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[21] E. L. Yip, "A note on the stability of solving a rank-p modification of a linear system by the Sherman-Morrison-Woodbury formula," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 2, pp. 507–513, 1986.