2013 European Control Conference (ECC)
July 17-19, 2013, Zürich, Switzerland.

On efficient predictive control of linear systems subject to quadratic constraints using condensed, structure-exploiting interior point methods

Markus Kögel and Rolf Findeisen

Abstract— **We describe a primal-dual interior point method tailored for predictive control of linear systems subject to convex quadratic costs and constraints. In particular, we consider only the inputs as decision variables to reduce the memory demand. Moreover, we propose to exploit the problem structure, which decreases the computational burden for long horizons. For the implementation we consider tailored solution methods of the arising Riccati recursion and the line-search. Furthermore, we discuss the memory and computational demand and problem specific methods to further save computations. Finally, we outline with an example the applicability.**

I. Introduction

In model predictive control (MPC) the feedback is given by the solution of an optimal control problem, which allows to directly consider constraints [16], [20]. In certain cases, the solution can be determined offline and stored [2], [3], which significantly reduces the amount of online computations. If the optimal control problem is a convex quadratic program, then its solution is a set of piece-wise affine control laws defined over polytopes. However, due the rapidly memory demand, explicit MPC is limited to small or medium size systems, cf. [2].

Therefore, in general the optimal control problem needs to be solved online, which is a challenge considering low-cost hardware, large systems or fast dynamics. MPC-tailored optimization algorithms can reduce the computational burden.

Different methods for online solution of the linear MPC problems have been investigated. Several works use interior point methods, where setting up and solving a system of linear equalities in each iteration is the major effort [19], [25]. Often the states and inputs are used as optimization variables, cf. [1], [6], [21], [24], [27] resulting in sparse linear systems. The system can be solved using for example structured Cholesky factorizations [6], [24], direct sparse elimination [27], [26] or Riccati recursions [1], [21]. Using only the states as decision variables results also in a sparse problem as outlined in [17]. [10] uses a closed-loop prediction the input is given by a linear state feedback plus a new input as decision variable. The linear controller enforces a finite impulse response such that for large enough horizon the linear system is banded.

Also other methods have been tailored for MPC, such as active set methods [8], first order methods [4], [13], [14], [22] or inexact interior point methods [23].

M. Kögel and R. Findeisen are with the Institute for Automation Engineering, Otto-von-Guericke-University Magdeburg, Magdeburg, Germany. {markus.koegel, rolf.findeisen}@ovgu.de.

The authors acknowledge the support by the International Max Planck Research School Magdeburg, Germany and the German Research Foundation.

In this work we use a primal-dual interior point method featuring only the inputs as decision variables and exploiting the problem structure. The key idea is to use the state trajectory, given through the dynamic of the plant, the input and measured state as auxiliary variables for the necessary computations. For the implementation we consider different methods to solve the Riccati recursion, line-search and further methods to avoid some calculations. Moreover, we discuss the memory and computational demand, to outline the reduced memory demand compared to other method and the competitive computational complexity.

In the remainder of this work, we first outline the problem setup in Section II and review primal-dual interior point methods in Section III. Section IV and V contain the main results: Section IV outlines how to utilize the problem structure and Section V presents important implementation aspects. In Section VI we present an example. Section VII discusses an extension to primal-barrier methods. Section VIII provides a summary and outlines future research directions.

The notation is standard. We denote by $v_1 \circ v_2$ the Hadamard product of the vectors v_1, v_2 and by $v[i]$ the ith entry of v. For square M $M > 0$ ($M \geq 0$) means that M is symmetric and positive (semi) definite. For $M > 0$, $M^{\frac{1}{2}}$ is such that $(M^{\frac{1}{2}})' M^{\frac{1}{2}} = M$ and $M^{\frac{1}{2}}$ is upper triangular.

II. Problem setup

We consider discrete-time linear systems of the form

$$\hat{x}_{k+1} = A_k \hat{x}_k + B_k \hat{u}_k + d_k, \tag{1}$$

which can be time-varying. $\hat{x}_k \in \mathbb{R}^{n_i}$ denotes the state, $\hat{u}_k \in \mathbb{R}^{p_i}$ the input and $d_k \in \mathbb{R}^{n_i}$ an affine term, e.g. a known disturbance. We assume that $n_i \geq 1$, $p_i \geq 0$ and that the matrices have the appropriate dimensions, see [12].

We use a horizon of length N, so we need to determine an input sequence $\mathbf{u} = \left(u_k', \ldots, u_{k+N}'\right)'$ and predicted state trajectory $\mathbf{x} = \left(x_k', \ldots, x_{k+N}'\right)'$ such that they are consistent with the dynamics (1) and the measured state $\hat{x}_k$, satisfy constraints and minimize a performance index. For consistency $\mathbf{u}$ and $\mathbf{x}$ need to satisfy

$$x_k = \hat{x}_k, \quad x_{i+1} = A_i x_i + B_i u_i + d_i,\ i = k, \ldots, k+N-1. \tag{2}$$

Using $\mathcal{A}_i$, $\mathcal{B}_i$, $\mathcal{A}_i^d$ as defined in [12] we can express the consistency conditions in terms of the measured state $\hat{x}_k$, the input vector $\mathbf{u}$ and affine terms $\mathbf{d}_k = \left(d_k', \ldots, d_{k+N-1}'\right)'$

$$x_i = \mathcal{A}_i \hat{x}_k + \mathcal{B}_i \mathbf{u} + \mathcal{A}_i^d \mathbf{d}_k. \tag{3}$$

We consider a convex performance index J given by

$$J = \sum_{i=k}^{k+N-1} \frac{1}{2} \begin{pmatrix} x_i \\ u_i \end{pmatrix}' \begin{pmatrix} Q_i & V_i \\ V_i' & R_i \end{pmatrix} \begin{pmatrix} x_i \\ u_i \end{pmatrix} + \begin{pmatrix} q_i \\ r_i \end{pmatrix}' \begin{pmatrix} x_i \\ u_i \end{pmatrix} \quad (4)$$
$$\begin{pmatrix} Q_i & V_i \\ V_i' & R_i \end{pmatrix} = \begin{pmatrix} Q_i & V_i \\ V_i' & R_i \end{pmatrix}' \geq 0.$$

With respect to the constraints we assume that there are m_i convex, quadratic constraints $f_{i,j} \leq 0$ for stage i

$$f_{i,j} = \frac{1}{2} \begin{pmatrix} x_i \\ u_i \end{pmatrix}' E_{i,j} \begin{pmatrix} x_i \\ u_i \end{pmatrix} + \begin{pmatrix} C_{i,j} \\ D_{i,j} \end{pmatrix}' \begin{pmatrix} x_i \\ u_i \end{pmatrix} - e_{i,j} \quad (5)$$
$$E_{i,j} = \begin{pmatrix} X_{i,j} & Y_{i,j} \\ Y_{i,j}' & U_{i,j} \end{pmatrix} = E_{i,j}' \geq 0,$$

i.e., we have in total $\mathcal{N}^c = \sum_{i=k}^{k+N} m_i$ constraints.

We consider only the input sequence $\mathbf{u}$ as optimization variable: the state trajectory $\mathbf{x}$ is given by the measured state $\hat{x}_k$ and the input trajectory $\mathbf{u}$ through the relationship (2). This results in the condensed constraints $f_{i,j}^c(\mathbf{u}, \hat{x}_k) = f_{i,j}(u_i, \mathcal{A}_i \hat{x}_k + \mathcal{B}_i u_k)$ given by

$$f_{i,j}^c(\mathbf{u}, \hat{x}_k) = \frac{1}{2} \|M_{i,j}\mathbf{u}\|_2^2 + G_{i,j}(\hat{x}_k)\mathbf{u} - g_{i,j}(\hat{x}_k) \quad (6a)$$
$$f_i^c(\mathbf{u}, \hat{x}_k) = \left(f_{i,1}^c(\mathbf{u}, \hat{x}_k)', \ldots, f_{i,m_i}^c(\mathbf{u}, \hat{x}_k)'\right)' \quad (6b)$$
$$\mathbf{f}^c(\mathbf{u}, \hat{x}_k) = \left(f_k^c(\mathbf{u}, \hat{x}_k)', \ldots, f_{k+N}^c(\mathbf{u}, \hat{x}_k)'\right)', \quad (6c)$$

where $M_{i,j}$, $G_{i,j}$ and $g_{i,j}$ are as in [12]. The condensed cost function with $\mathbf{H}$ and $\mathbf{h}(\hat{x}_k)$ as in [12] is given by

$$J^c = \frac{1}{2}\mathbf{u}'\mathbf{H}\mathbf{u} + \mathbf{u}'\mathbf{h}(\hat{x}_k) + \text{const.} \quad (7)$$

Summarizing, for predictive control of the plant (1) subject to the performance index (4) and constraints (5) the feedback $\hat{u}_k = u_k$ is the solution of the optimization problem

$$\min_{\mathbf{u}} \frac{1}{2}\mathbf{u}'\mathbf{H}\mathbf{u} + \mathbf{u}'\mathbf{h}(\hat{x}_k) \text{ subject to } \mathbf{f}^c(\mathbf{u}, \hat{x}_k) \leq 0, \quad (8)$$

which is in general a quadratically constrained quadratic program. In case there are no quadratic constraints (all $E_{i,j} = 0$) it is a quadratic program. It is a linear program if also the cost function is affine (all $E_{i,j}$, Q_i, R_i, V_i are zero).

Note that this setup is rather general: We can handle quadratic constraints such as ellipsoidal terminal constraints [18], or power constraints, see [6]. Moreover, we allow time-dependent dimensions appearing in periodic systems [9]. Also control setups with a terminal input ($p_{k+N} > 0$) are possible, e.g. parametrized terminal constraints [15].

In the remaining paper we outline the solution of (8) using structure exploiting interior point methods.

III. Primal-dual interior point methods

This work considers online optimization of (8) using primal-dual interior point methods, which we review in the following. We refer for details on interior point methods to [19], [25]. The following mild assumption guarantees that each iteration of the interior point method is well defined.

Assumption 1: (Regularity of (8))
For each $\mathbf{u}$ there exists a $\eta > 0$ such that

$$\mathbf{H} + \sum_{i=k}^{k+N} \sum_{j=1}^{m_i} \nabla_{\mathbf{u}}^2 f_{i,j}^c + \nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k)' \nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k) \geq \eta.$$

Note that this assumption is satisfied if all $R_i > 0, \forall i$ or the input is bounded.

In primal-dual interior point methods one determines a solution to (8) by iteratively improving the input sequence $\mathbf{u}$, slack variables $\mathbf{s} > \mathbf{0}$ and Lagrange multipliers $\boldsymbol{\lambda} > \mathbf{0}$[1]. This is done by minimizing the residuals of perturbed KKT conditions, cf. [19], [25]

$$\boldsymbol{\zeta}^p = \mathbf{f}^c(\mathbf{u}, \hat{x}_k) + \mathbf{s} \quad (9a)$$
$$\boldsymbol{\zeta}^d = \mathbf{H}\mathbf{u} + \mathbf{h}(\hat{x}_k) + \nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k)'\boldsymbol{\lambda} \quad (9b)$$
$$\boldsymbol{\theta} = \boldsymbol{\lambda} \circ \mathbf{s} - \rho\mu\mathbf{1}, \quad (9c)$$

using Newton's method. In (9) $\boldsymbol{\zeta}^p$ denotes the primal and $\boldsymbol{\zeta}^d$ the dual residual, $\mu = \frac{\mathbf{s}'\boldsymbol{\lambda}}{\mathcal{N}^c}$ is a complementary measure and $\rho \in [0, 1)$ is the centering parameter. In each iteration

$$\mathbf{L} \begin{pmatrix} \Delta\mathbf{u} \\ \Delta\mathbf{s} \\ \Delta\boldsymbol{\lambda} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\zeta}^d \\ \boldsymbol{\zeta}^p \\ \boldsymbol{\theta} \end{pmatrix} = 0, \quad (10)$$

is solved to obtain the search direction $\Delta\mathbf{u}$, $\Delta\mathbf{s}$, $\Delta\boldsymbol{\lambda}$, where

$$\mathbf{L} = \begin{pmatrix} \nabla_{\mathbf{u}}^2 \mathcal{L}(\boldsymbol{\lambda}) & 0 & +\nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k)' \\ \nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k) & I & 0 \\ 0 & \boldsymbol{\Lambda} & \mathbf{S} \end{pmatrix}, \quad (11)$$

and $\mathbf{S} = \text{diag}(\mathbf{s})$, $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda})$. $\nabla_{\mathbf{u}}^2 \mathcal{L}(\boldsymbol{\lambda}, \hat{x}_k)$ is the second derivative of the Lagrangian with respect to $\mathbf{u}$ and given by

$$\nabla_{\mathbf{u}}^2 \mathcal{L}(\boldsymbol{\lambda}) = \mathbf{H} + \sum_{i=k}^{k+N} \sum_{j=1}^{m_i} \lambda_{i,j} \nabla_{\mathbf{u}}^2 f_{i,j}^c = \mathbf{H} + \mathbf{F}(\boldsymbol{\lambda}). \quad (12)$$

Note that (10) has a unique solution, because $\mathbf{L}$ is regular due to Assumption 1, compare Remark 1.

Remark 1: (Solution of linear system (10))
In this work we solve the problem (10) using so-called normal-equations, cf. [19]: first one determines $\Delta\mathbf{u}$ from

$$\tilde{\mathbf{L}}\Delta\mathbf{u} + \tilde{\mathbf{l}} = 0, \quad (13)$$

where

$$\tilde{\mathbf{L}} = \mathbf{H} + \mathbf{F}(\boldsymbol{\lambda}) + \nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k)'\mathbf{W}\nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k) \quad (14a)$$
$$\tilde{\mathbf{l}} = \boldsymbol{\zeta}^d + \nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k)'(\mathbf{W}\boldsymbol{\zeta}^p - \mathbf{S}^{-1}\boldsymbol{\theta}), \quad (14b)$$

and $\mathbf{W} = \mathbf{S}^{-1}\boldsymbol{\Lambda}$. Then one computes $\Delta\boldsymbol{\lambda}$ and $\Delta\boldsymbol{s}$ using

$$\Delta\boldsymbol{s} = -\nabla_{\mathbf{u}}\mathbf{f}^c(\mathbf{u}, \hat{x}_k)\Delta\mathbf{u} - \boldsymbol{\zeta}^p \quad (15a)$$
$$\Delta\boldsymbol{\lambda} = -\mathbf{W}\Delta\boldsymbol{s} - \mathbf{S}^{-1}\boldsymbol{\theta}. \quad (15b)$$

Due to Assumption 1 and since $\lambda_{i,j} > 0$, $\mathbf{W} > 0$

$$\tilde{\mathbf{L}} \geq \min(\min(\lambda_{i,j}), \min(W_{i,j}), 1)\eta > 0. \quad (16)$$

[1]We denote by $\mathbf{u}$, $\mathbf{s}$, $\boldsymbol{\lambda}$ the current iterate of the interior point method and by $\mathbf{u}^+$, $\mathbf{s}^+$, $\boldsymbol{\lambda}^+$ the next iterate.

Thus $\tilde{\mathbf{L}}$ is nonsingular. Moreover, solving (10) directly or via (14), (15) is equivalent, i.e. also $\mathbf{L}$ is regular [19].

Remark 2: (Choice of stepsize)
To obtain the new iterates a stepsize $\beta \in [0,1]$ is used

$$\mathbf{u}^+ = \mathbf{u} + \beta\Delta\mathbf{u} \quad \mathbf{s}^+ = \mathbf{s} + \beta\Delta\mathbf{s} \quad \boldsymbol{\lambda}^+ = \boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}. \quad (17)$$

If there are no quadratic constraints, then one can use as stepsize β the maximum α satisfying

$$\mathbf{s} + \alpha\Delta\mathbf{s} \geq (1-\tau)\mathbf{s} \qquad \boldsymbol{\lambda} + \alpha\Delta\boldsymbol{\lambda} \geq (1-\tau)\boldsymbol{\lambda}, \qquad (18)$$

where $\tau \in (0,1)$ is a tuning parameter, cf. [19, p. 483].

For quadratic constraints the stepsize β need to be small enough such that $\mathbf{u}^+$, $\mathbf{s}^+$, $\boldsymbol{\lambda}^+$ reduce the norm of the residuals (9), see [19], [25]. Often backtracking is used: one starts with a trial stepsize $\gamma = \alpha$ and reduces it further by multiplying it with some $\omega \in (0,1)$, until[2]

$$\left\| \begin{pmatrix} (\boldsymbol{\zeta}^d)^+ \\ (\boldsymbol{\zeta}^p)^+ \\ \boldsymbol{\theta}^+ \end{pmatrix} \right\| \leq \xi \left\| \begin{pmatrix} \boldsymbol{\zeta}^d \\ \boldsymbol{\zeta}^p \\ \boldsymbol{\theta} \end{pmatrix} \right\| \qquad \xi = (1-\sigma\gamma) \qquad (19)$$

for some $\sigma \in (0,1)$. This line-search might require several iterations and thus evaluations of (9). Fortunately, a lower bound on γ can be determined as follows, c.f. [12].

Proposition 1: (Lower bound on stepsize γ)
Any $\gamma \in [0,\alpha]$ such that

$$\gamma \left\| \begin{pmatrix} \boldsymbol{\kappa} \\ \boldsymbol{\upsilon} \\ \boldsymbol{\chi} \end{pmatrix} \right\| \leq (1-\sigma) \left\| \begin{pmatrix} \boldsymbol{\zeta}^d \\ \boldsymbol{\zeta}^p \\ \boldsymbol{\theta} \end{pmatrix} \right\| \qquad (20a)$$

$$\upsilon_{i,j} = 0.5\Delta\mathbf{u}' M_{i,j} \Delta\mathbf{u} \qquad (20b)$$

$$\boldsymbol{\kappa} = \sum_{i=k}^{k+N} \sum_{j=1}^{m_i} \Delta\lambda_{i,j} M_{i,j} \Delta\mathbf{u} \qquad \boldsymbol{\chi} = \Delta\mathbf{s} \circ \Delta\boldsymbol{\lambda}, \qquad (20c)$$

satisfies the condition (19)*, where $\boldsymbol{\upsilon}$ is given by $\upsilon_{i,j}$ similar as* (6)*. There exist $\gamma > 0$ unless $\boldsymbol{\zeta}^p = 0$, $\boldsymbol{\zeta}^d = 0$ and $\boldsymbol{\theta} = 0$.*
This result can be used as alternative to backtracking as follows: compute $\boldsymbol{\kappa}$, $\boldsymbol{\upsilon}$, $\boldsymbol{\chi}$ and determine the maximum $\gamma \leq \alpha$ from (20a). This directly delivers as suitable setpsize $\beta = \gamma$, but γ might be small. So it seems useful, to try a fixed number of stepsizes between α and γ to further decrease $(\boldsymbol{\zeta}^d)^+$, $(\boldsymbol{\zeta}^p)^+$ and $\boldsymbol{\theta}^+$. This approach guarantees a maximum run-time of the line-search and that the residuals decrease.

IV. Exploiting the structure of the MPC setup

Next we outline a special approach that exploits the structure of the MPC problem, which can be advantageous as discussed in the next section. The computational effort of the interior point method presented above is dominated by

1) the evaluation of constraints $\mathbf{f}^c(\mathbf{u}, \hat{x}_k)$ in (9a)
2) the computation of the dual residual $\boldsymbol{\zeta}^d$ (9b)
3) the setup and solution of linear systems (13)
4) the computation of $\nabla_{\mathbf{u}} \mathbf{f}^c(\mathbf{u}, \hat{x}_k)\Delta\mathbf{u}$ in (15a),

since they require additions and multiplications involving the matrices $\mathbf{H}$, $G_{i,j}$, $M_{i,j}$ and the solution of (13). In the works [6], [10], [17], [21], [24] the problem structure is exploited using sparsity. Here the matrices $\mathbf{H}$, $G_{i,j}$, $M_{i,j}$ are in general dense, but it is also here possible to exploit the problem structure such that only computations with the (smaller) matrices in (1), (4) and (5) are required.

We refer to Section V for methods to use further structures in the dynamics, cost functions and constraints as well as a detailed discussion and comparison.

1) Evaluation of constraints $\mathbf{f}^c(\mathbf{u}, \hat{x}_k)$:
To evaluate $\mathbf{f}^c(\mathbf{u}, \hat{x}_k)$ efficiently, let us assume we know the state trajectory $\mathbf{z}$ induced by $\hat{x}_k$, $\mathbf{u}$, $\mathbf{d}_k$ and (3), i.e.

$$z_k = \hat{x}_k, \quad z_{i+1} = A_i z_i + B_i u_i + d_i, \qquad (21)$$

where $i = k, \ldots, k+N-1$. In this case we can compute $\mathbf{f}^c(\mathbf{u}, \hat{x}_k)$ using (9a) element-wise as

$$f^c_{i,j} = \frac{1}{2} \| E_{i,j} \begin{pmatrix} z_i \\ u_i \end{pmatrix} \|_2^2 + \begin{pmatrix} C_{i,j} \\ D_{i,j} \end{pmatrix}' \begin{pmatrix} z_i \\ u_i \end{pmatrix} - e_{i,j}.$$

We can compute $\mathbf{z}$ by forward iterating (21). However this is only necessary for the first iteration, since we can use the update scheme (31) defined below to save computations.

2) Dual residual $\boldsymbol{\zeta}^d$:
Note that the first two parts of $\boldsymbol{\zeta}^d$ (9b) depend on the cost function (7) and the last part on the constraints (6). They can be evaluated from $\mathbf{u}$, $\mathbf{z}$ using a backward iteration, see e.g. [5, Ch 1.9.] for the first parts and e.g. [13] for the last part.
Combining both iterations results in, see [12]

$$\zeta^d_i = B_i'\delta_{i+1} + R_i u_i + r_i + V_i' z_i + \sum_{j=1}^{m_i} \tilde{D}_{i,j}\lambda_i \qquad (22a)$$

$$\delta_i = A_i'\delta_{i+1} + Q_i z_i + q_i + V_i u_i + \sum_{j=1}^{m_i} \tilde{C}_{i,j}\lambda_i, \qquad (22b)$$

with $\delta_{k+N+1} = 0$ and $i = k+N, \ldots, k$. $\tilde{C}_{i,j}$ and $\tilde{D}_{i,j}$ correspond to the linearization of the constraints around the state trajectory $\mathbf{z}$ (21) and input sequence $\mathbf{u}$ (see [12])

$$\tilde{C}_{i,j} = C_{i,j} + X_{i,j} z_i + Y_{i,j} u_i \qquad (23a)$$

$$\tilde{D}_{i,j} = D_{i,j} + Y_{i,j}' z_i + U_{i,j} u_i. \qquad (23b)$$

3) Setup and solution of the linear system (13)*:*
We outline how the problem structure can be used to setup and solve (13). First we show that the solution of (13) correspond to the solution of an unconstrained optimal control problems with similar structure as the MPC problem (8)

Proposition 2: (Equivalence of solution)
$\Delta\mathbf{u}$ is a solution of the systems of linear equalities (13)*, if and only if, it is a solution of the unconstrained, quadratic optimal control problems*

$$\min_{\Delta\mathbf{u}} \tilde{J} \qquad (24)$$

with

$$\tilde{J} = \sum_{i=k}^{k+N} \frac{1}{2} \Delta\mathbf{u}' (\mathcal{B}_i' \tilde{Q}_i \mathcal{B}_i + 2\mathcal{B}_i' \tilde{V}_i \mathcal{Z}_i + \mathcal{Z}_i' \tilde{R}_i \mathcal{Z}_i) \Delta\mathbf{u} + \sum_{i=k}^{k+N} \Delta\mathbf{u}' (\mathcal{B}_i' \tilde{q}_i + \mathcal{Z}_i' \tilde{r}_i) \qquad (25)$$

[2] $(\boldsymbol{\zeta}^p)^+$, $(\boldsymbol{\zeta}^d)^+$ and $\boldsymbol{\theta}^+$ denote (9) with the new iterate $\mathbf{u}^+$, $\mathbf{s}^+$, $\boldsymbol{\lambda}^+$. Notice that $\boldsymbol{\theta}^+$ depends still on the old μ.

and where

$$\tilde{Q}_i = Q_i + \sum_{j=1}^{m_i} \lambda_{i,j} X_{i,j} + W_{i,j}\tilde{C}_{i,j}\tilde{C}'_{i,j} \tag{26a}$$

$$\tilde{R}_i = R_i + \sum_{j=1}^{m_i} \lambda_{i,j} U_{i,j} + W_{i,j}\tilde{D}_{i,j}\tilde{D}'_{i,j} \tag{26b}$$

$$\tilde{V}_i = V_i + \sum_{j=1}^{m_i} \lambda_{i,j} Y_{i,j} + W_{i,j}\tilde{C}_{i,j}\tilde{D}'_{i,j} \tag{26c}$$

$$\tilde{q}_i = \sum_{j=1}^{m_i} \tilde{C}_{i,j}(W_{i,j}\zeta^p_{i,j} - S^{-1}_{i,j}\theta_{i,j}) \tag{26d}$$

$$\tilde{r}_i = \zeta^d_i + \sum_{j=1}^{m_i} \tilde{D}_{i,j}(W_{i,j}\zeta^p_{i,j} - S^{-1}_{i,j}\theta_{i,j}). \tag{26e}$$

This proposition is verified in [12].

The problem (24) has a similar structure as (8), but no constraints, the weights (26) instead of (4) and $\hat{x}_k = 0$, $\mathbf{d}_k = 0$. One can solve (24) using Riccati iterations, see [7].

Corollary 1: (Riccati based solution of (24)*, [7])*
If (24) *has a solution* $\Delta\mathbf{u}$*, then it satisfies*

$$\Delta u_i = -\Psi_i^{-1}(\Theta_i \Delta z_i + \tilde{r}_i + B'_i\xi_{i+1}) \tag{27a}$$

where Δz_i *for* $i = k, \dots, k+N$ *is given by*

$$\Delta z_{i+1} = A_i\Delta z_i + B_i\Delta u_i \qquad \Delta z_k = 0. \tag{27b}$$

The terms $\tilde{r}_i + B'_i\xi_{i+1}$*,* Ψ_i^{-1} *and* Θ_i *are given by*

$$P_i = A'_iP_{i+1}A_i + \tilde{Q}_i - \Theta'_i\Psi_i^{-1}\Theta_i \tag{28a}$$

$$\Psi_i = \tilde{R}_i + B'_iP_{i+1}B_i, \quad \Theta_i = \tilde{V}_i' + B'_iP_{i+1}A_i \tag{28b}$$

$$\xi_i = \tilde{q}_i + A'_i\xi_{i+1} - \Theta'_i\Psi_i^{-1}(\tilde{r}_i + B'_i\xi_{i+1}) \tag{29}$$

where $i = k+N, \dots, k$*,* $P_{k+N+1} = 0$ *and* $\xi_{k+N+1} = 0$*.*

Note that this approach consists of a backward iteration (28), (29) followed by a forward iteration (27). Since we assume that $\tilde{\mathbf{L}}$ is regular Ψ_i^{-1} exists, cf. [7]. Moreover, it features only smaller matrices from (1), (4) and (5).

4) Computation of $\nabla_{\mathbf{u}} f(\mathbf{u}; \hat{x}_k)\Delta\mathbf{u}$*:*
Finally, we need $\nabla_{\mathbf{u}} f(\mathbf{u}; \hat{x}_k)\Delta\mathbf{u}$ for (15). To exploit the problem structure note that using (21) we have (compare [12])

$$\begin{aligned} \nabla_{\mathbf{u}} f(\mathbf{u}; \hat{x}_k)\Delta\mathbf{u} &= \langle y_{k,1}, \dots, y_{k+N,m_{k+N}} \rangle \\ y_{i,j} &= \tilde{C}'_{i,j}\Delta z_i + \tilde{D}'_{i,j}\Delta u_i, \end{aligned} \tag{30}$$

where $\Delta u_i, \Delta z_i$ have already been computed in the solution of (13), cf. (27). One can use $\Delta\mathbf{z}$ from (27) to compute

$$\begin{aligned} z_i^+ &= \mathcal{A}_i\hat{x}_k + \mathcal{B}_i\mathbf{u}^+ + \mathcal{A}_d^i\mathbf{d}_k \\ &= \underbrace{\mathcal{A}_i\hat{x}_k + \mathcal{B}_i\mathbf{u} + \mathcal{A}_i^d\mathbf{d}_k}_{z_i} + \beta\underbrace{\mathcal{B}_i\Delta\mathbf{u}}_{\Delta z_i}, \end{aligned} \tag{31}$$

compare [12], needed in the next iteration. This update avoids to compute $\mathbf{z}^+$ using (21).

Remark 3: (Interpretation of structure exploiting)
If we would use $\mathbf{u}$ and $\mathbf{x}$ as decision variable such as in e.g. [21], then we would have additional primal / dual variables: $\mathbf{x}$ and $\boldsymbol{\mu}$, the Lagrange multiplier related to the equality constraints (2). In a nutshell we enforce that $\mathbf{x}$ satisfies (2), which results in $\mathbf{z}$ (21) parametrized by $\mathbf{u}$ and $\hat{x}_k$. Moreover, with the Lagrange multiplier $\boldsymbol{\mu}$ as $\boldsymbol{\delta}$ (22b) we guarantee that $\nabla_{\mathbf{x}}\overline{\mathcal{L}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = 0$, where is $\overline{\mathcal{L}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ the Lagrangian for this case.

V. Implementation and computational effort

In this section we first outline the impact of the constraints on the computational effort and then discuss the solution of the Riccati recursion and its computational effort. Afterwards, we discuss the memory demand, extensions and compare the algorithm with other methods.

A. Computational aspects related to the constraints

The constraints influence the computational effort. Therefore we denote in the following by $\mathcal{F}^E_{i,j}$ the FLOP count[3] to compute

$$E_{i,j}\begin{pmatrix} z_i \\ u_i \end{pmatrix} = \begin{pmatrix} X_{i,j} & Y_{i,j} \\ Y'_{i,j} & U_{i,j} \end{pmatrix}\begin{pmatrix} z_i \\ u_i \end{pmatrix}, \tag{32}$$

which is required to evaluate the primal residual $\boldsymbol{\zeta}^p$ (22a) and to linearize the constraints (23). $\mathcal{F}^W_{i,j}$ is the FLOP count to compute the quadratic weightings in (26). The overall FLOP count due to the constraints in an iteration is given by:

$$\mathcal{F}^{Const.} = \sum_{i=k}^{k+N}\sum_{j=1}^{m_i} \mathcal{F}^W_{i,j} + \#_{RE}\mathcal{F}^E_{i,j}, \tag{33}$$

where $\#_{RE}$ is the number of times the residuals (9a), (9b) are evaluated per iteration. If there are no quadratic constraints, then $\#_{RE} = 1$: we only need to evaluate (18), else $\#_{RE}$ depends on the actual line-search method.

Let us consider different constraint types, to save additional computations, see Table I for the FLOP counts.

1) General constraints:
In general the matrices $C_{i,j}$, $D_{i,j}$, $E_{i,j}$ (5) have no special structure. Using the symmetry of $\tilde{Q}_i$ / $\tilde{R}_i$ saves computations.

2) Affine constraints:
Affine constraints have $E_{i,j} = 0$, i.e., we do not need to evaluate (32). Also the computational effort of (26) is halved, since there are no quadratic terms ($U_{i,j}$, $X_{i,j}$, $Y_{i,j}$ are zero).

3) Two sided affine constraints:
A two sided affine constraints is given by two affine constraints $f_{i,j} \le 0$, $f_{i,l} \le 0$ with $C_{i,j} = -C_{i,l}$, $D_{i,j} = -D_{i,l}$:

$$f_{i,j} \le 0, f_{i,l} \le 0 \Leftrightarrow -e_{i,j} \le \begin{pmatrix} C_{i,j} \\ D_{i,j} \end{pmatrix}' \begin{pmatrix} x_i \\ u_i \end{pmatrix} \le e_{i,l}, \tag{34}$$

We have $W_{i,j}C_{i,j}C'_{i,j} + W_{i,l}C_{i,l}C'_{i,l} = (W_{i,j} + W_{i,l})C_{i,j}C'_{i,j}$ for $\tilde{Q}_i$ (and similarly $\tilde{V}_i$, $\tilde{R}_i$), which saves FLOPs.

4) Pure state / input constraints :
For pure state constraints we have $C_{i,j}D'_{i,j} = 0$, $D_{i,j} = 0$, $U_{i,j} = 0$ and $Y_{i,j} = 0$, i.e., we can avoid the computations with them (similarly for pure input constraints). Note that combinations with 2) or 3) are possible.

5) Bound constraints:
For bounds on the state or input, e.g. $x_i[1] < 2$, $C_{i,j}$, $D_{i,j}$ are extremely simple (contain maximally one non-zero entry) and $E_{i,j} = 0$, which reduces the computational effort to a constant term per constraint.

In summary, exploiting special structure can have a significant impact on the computations. Moreover, the evaluation

[3]The FLOP count counts the floating point operations (FLOPS): A FLOP is a addition, multiplication or division of two floating numbers or taking one square root. Here we focus on the leading terms for simplicity.

TABLE I
FLOP COUNTS $\mathcal{F}_{i,j}^E$, $\mathcal{F}_{i,j}^W$ FOR A CONSTRAINT $f_{i,j} \leq 0$

Constraint type	$\mathcal{F}_{i,j}^E$	$\mathcal{F}_{i,j}^W$
1) General constraint	$2(n_i+p_i)^2$	$2(n_i+p_i)^2$
2) Affine constraint	0	$(n_i+p_i)^2$
3) Two sided affine constraint	0	$0.5(n_i+p_i)^2$
4) Pure state / input constraint	$2n_i^2/2p_i^2$	$2n_i^2/2p_i^2$
Combination 2) + 4)	0	n_i^2 / p_i^2
Combination 3) + 4)	0	$0.5n_i^2$ / $0.5p_i^2$
5) Bound on state / input	0	$O(1)$

of the primal residual (22a) and the linearization of the constraints (23) for different time instances can be performed easily in parallel.

B. Solution of the Riccati recursion (28)

The solution of the Riccati recursion (28) has a major impact on the computational demand. Therefore let us discuss different approaches to solve it. In detail, we consider two types of methods: propagating P_i and *array algorithms*, which propagate the Cholesky factor $P_i^{\frac{1}{2}}$, see [11]. Array algorithms have better numerical properties [11], e.g. reduced round off errors and $(P_i^{\frac{1}{2}})'P_i^{\frac{1}{2}}$ is guaranteed to be positive semi-definite even for significant numerical errors or badly scaled problems, but it might be slower in practice. Finally, we outline a method to use parallel computation to allow a further speedup of the computations.

1) Propagating P_i :

We shortly discuss the solution of (28) and refer for more details to [12]. For the computation of $\Theta_i'\Psi_i^{-1}\Theta_i$ we determine first a Cholesky factor $\Psi_i^{\frac{1}{2}}$ of Ψ_i, then compute $\Theta_i'(\Psi_i^{\frac{1}{2}})^{-1}$ via Cholesky matrix substitution, then we compute $\Theta_i'\Psi_i^{-1}\Theta_i$, where we exploit symmetry to save calculations. We can compute $A_i'P_{i+1}A_i$, $B_i'P_{i+1}B_i$, $B_i'P_{i+1}A_i$ (for Θ_i) either using matrix-matrix multiplications

$$A_i'P_{i+1}A_i = A_i' \cdot (P_{i+1}A_i) \tag{35a}$$
$$B_i'P_{i+1}B_i = B_i' \cdot (P_{i+1}B_i) \tag{35b}$$
$$B_i'P_{i+1}A_i = B_i' \cdot (P_{i+1}A_i) \tag{35c}$$

exploiting, where possible, symmetries and caching $(P_{i+1}A_i)$ for $B_i'P_{i+1}A_i$. Alternatively we compute first a Cholesky factor P_{i+1} and then evaluate $A_i'P_{i+1}A_i$, $B_i'P_{i+1}B_i$, $B_i'P_{i+1}A_i$ as

$$A_i'P_{i+1}A_i = (P_{i+1}^{\frac{1}{2}}A_i)' \cdot (P_{i+1}^{\frac{1}{2}}A_i) \tag{36a}$$
$$B_i'P_{i+1}B_i = (P_{i+1}^{\frac{1}{2}}B_i)' \cdot (P_{i+1}^{\frac{1}{2}}B_i) \tag{36b}$$
$$B_i'P_{i+1}A_i = (P_{i+1}^{\frac{1}{2}}B_i)' \cdot (P_{i+1}^{\frac{1}{2}}A_i) \tag{36c}$$

using again symmetries and caching terms where possible.

In Table II (36) the FLOP count $\mathcal{F}_i^{Riccati}$ of a single evaluation of the Riccati recursion is illustrated, where we assume for simplicity a constant state dimensions. We observe that (35) has a larger FLOP count than (36), however (35) can exploit sparsity of A_i, B_i better as outlined below.

Note that here we avoid to compute Ψ_i^{-1}, which requires minor adaptions to (27), (29), cf. [12].

Now consider $\tilde{V}_i = 0$, which means that there is no cross-weighting term ($V_i = 0$) and there are only pure state constraints/input constraints (see Section (V-A.4)). In this case we can split the recursion (28) into two parts, cf. [11], which reduces the computational effort further (for details, see [12]. We have

$$\overline{P}_{i+1} = P_{i+1} - P_{i+1}B_i'\Psi_i^{-1}B_iP_{i+1} \tag{37a}$$
$$P_i = A_i'\overline{P}_{i+1}A_i + \tilde{Q}_i. \tag{37b}$$

If in addition $\tilde{R}_i$ is diagonal, i.e., the weighting matrix R_i is diagonal and there are only bound constraints on the input, then we can simplify the evaluation of (28) further by *"sequentially updating the inputs"*, cf. [11]. In detail, we compute for $P_{i+1}^{(1)} = P_{i+1}$, $j = 1, \ldots, p_i$

$$\psi_i^{(j)} = B_i[j]'P_{i+1}^{(j)}B_i[j] + \tilde{R}_i[j,j] \tag{38a}$$
$$P_{i+1}^{(j+1)} = P_{i+1}^{(j)} - \frac{P_{i+1}^{(j)}B_i[j](B_i[j])'P_{i+1}^{(j)}}{\psi_i^{(j)}}, \tag{38b}$$

where $B_i[j]$ is the jth column of B_i, $R_i[j,j]$ is the jth entry on the diagonal of R_i and $P_{i+1}^{(p_i+1)} = \overline{P}_{i+1}$. Finally, we update P_i by

$$P_i = A_i'\overline{P}_{i+1}A_i + \tilde{Q}_i. \tag{38c}$$

Note that for both cases we can compute $A_i'\overline{P}_{i+1}A_i$ via (36a) to reduce the FLOP count. Again minor adaptions to (27), (29) are required as outlined in [12].

To sum up, employing approaches tailored to the structure of $\tilde{Q}_i$, $\tilde{V}_i$, $\tilde{R}_i$ helps to decrease the computational effort.

Remark 4: (Boundary cases $i = k$, $i = k+N$)
For $i = k+N$, we can use $P_{k+N+1} = 0$ to reduce the effort. In particular, if there is no terminal input ($p_{k+N} = 0$), then $P_{k+N} = Q_{k+N}$. Note that we do not need to compute P_k.

Remark 5: (Utilizing sparsity of A_i, B_i)
Sparsity or structure of A_i, B_i can speed up the computation.

In the time-invariant case one can choose a state transformation $\tilde{x} = Tx$ to enforce special structure into $\tilde{A}$. However trade-offs are necessary. Transforming the state could destroy the structure of the constraints, e.g. bound constraints become two-sided affine constraints. Moreover, if one transforms A into its block real Jordan form $\tilde{A}$, then $\tilde{A}$ has only $O(n)$ entries, which dramatically reduces the multiplications computational effort: for example the n^3 term in Table II vanishes if (35) is used. However the numerical reliability of the algorithm might decrease, since such a transformation is usually ill-conditioned. On the other hand using an orthogonal T is numerically reliable, but it can enforce only about $0.5n^2$ zeros into $\tilde{A}$, [11].

2) Solution of (28) *using array algorithm :*
Array algorithm provide usually a more accurate solution of (28), see [11]. In the following we present only the case $\tilde{V}_i = 0$ and refer for more details to [12].

First the so-called pre-array $\mathcal{P}^{pre}$ is set up

$$\mathcal{P}^{pre} = \begin{pmatrix} \tilde{R}_i^{\frac{1}{2}} & 0 \\ P_{i+1}^{\frac{1}{2}}B_i & P_{i+1}^{\frac{1}{2}}A_i \\ 0 & \tilde{Q}_i^{\frac{1}{2}} \end{pmatrix}, \tag{39a}$$

TABLE II

FLOP COUNT $\mathcal{F}_i^{Riccati}$ OF ONE EVALUATION OF RICCATI RECURSION (CONSTANT STATE DIMENSIONS $n = n_i$)

Method	$\mathcal{F}_i^{Riccati}$ (cubic terms)
(28), (35)	$3n^3 + 5n^2p_i + 2np_i^2 + \frac{1}{3}p_i^3$
(28), (36)	$\frac{7}{3}n^3 + 4n^2p_i + 2np_i^2 + \frac{1}{3}p_i^3$
(37), $A_i'\overline{P}_{i+1}A_i$ via (35a)	$3n^3 + 3n^2p_i + 2np_i^2 + \frac{1}{3}p_i^3$
(37), $A_i'\overline{P}_{i+1}A_i$ via (36a)	$\frac{7}{3}n^3 + 3n^2p_i + 2np_i^2 + \frac{1}{3}p_i^3$
(38), $A_i'\overline{P}_{i+1}A_i$ via (35a)	$3n^3 + 3n^2p_i$
(38), $A_i'\overline{P}_{i+1}A_i$ via (36a)	$\frac{7}{3}n^3 + 3n^2p_i$
Array algorithm, ($\tilde{V}_i \neq 0$), ([12])	$\frac{10}{3}n^3 + 6n^2p_i + 3np_i^2 + \frac{1}{3}p_i^3$
Array algorithm ($\tilde{V}_i = 0$) (39)	$\frac{10}{3}n^3 + 5n^2p_i + 2np_i^2 + \frac{1}{3}p_i^3$
$\tilde{V}_i = 0$, $\tilde{Q}_i$, $\tilde{R}_i$ diagonal	$3n^3 + 5n^2p_i + 2np_i^2$

which requires Cholesky decompositions to compute $\tilde{Q}_i^{\frac{1}{2}}$, $\tilde{R}_i^{\frac{1}{2}}$ and multiplications with the triangular $P_{i+1}^{\frac{1}{2}}$.

Second this pre-array is transformed using an orthogonal transformation T into an upper triangular matrix $\mathcal{P}^{post}$ the so-called post array, which satisfies

$$T\mathcal{P}^{pre} = \mathcal{P}^{post} = \begin{pmatrix} \Xi_i & \Omega_i \\ 0 & P_i^{\frac{1}{2}} \\ 0 & 0 \end{pmatrix}, \tag{39b}$$

where $\Psi_i = \Xi_i'\Xi_i$, $\Theta_i = \Xi_i'\Omega_i$, [11]. Since T is not needed explicitly, one can determine the post-array $\mathcal{P}^{post}$ efficiently by taking the structure of $\mathcal{P}^{pre}$ into account using for example Householder reflections, see [11]. Table II reports the dominant term of the FLOP count.

Note that with simple adaptions of (27) we can avoid the explicit computation of Ψ_i^{-1} and Θ_i, see [12].

Remark 6: (Reduced effort for special cases)
Also for the array algorithm certain cases reduce the computations. In Table II we present the FLOP count for the general case ($\tilde{V}_i \neq 0$) and another special case.

Moreover, sparsity of A_i, B_i can reduce the effort of the matrix multiplications and Householder reflections.

3) Parallelizing the Riccati recursion (28)*:*
Above we discussed the efficient solution of the Riccati recursion (28). Often more than one computation unit is available (e.g. for a multicore-processor), which can be exploited to improve the speed, see [14] and the references therein. To improve the speed we discuss a method to evaluate (28), which requires some additional computations, but allows a parallel evaluation. We focus on the case of fixed dimensions $n = n_i$, $p = p_i$ and no mixed input-state constraints and input-state weightings ($\tilde{V}_i = 0$).

In a nutshell we split the horizon N into η parts and solve (28) "independently" in each of the parts. Clearly, this is not directly possible, since to solve (28) correctly in a specific part, the correct end condition needs to be known. Therefore we propose the following procedure: first we solve (28) in each part with a zero end condition, then we use these solutions to compute for each part the correct end condition, which allows to solve in a third step (28) in each part correctly. Note that this method uses a known change of initial condition formula for Kalman filtering [11, Ch. 17.6] and the relationship between the Riccati recursion for the Kalman filter and LQR. For a simple presentation we assume $\eta\underline{N} = M + 1$, where $\underline{N}$ is the length of each part.

In the first step, we solve $i = j\underline{N}-1+k, \ldots, (j-1)\underline{N}+k$, $j = 2, \ldots, \eta$ with $P^I_{j\underline{N}+k} = 0$, $\mathcal{D}_{j\underline{N}+k} = I$, $\mathcal{O}_{j\underline{N}+k} = 0$:

$$P_i^I = A_i'P_{i+1}^IA_i + \tilde{Q}_i - (\Theta_i^I)'(\Psi_i^I)^{-1}\Theta_i^I \tag{40a}$$

$$\mathcal{D}_i = \mathcal{D}_{i+1}(I - B_i(\Psi_i^I)^{-1}B_i'P_{i+1}^I)A_i \tag{40b}$$

$$\mathcal{O}_i = \mathcal{O}_{i+1} + \mathcal{D}_{i+1}B_i(\Psi_i^I)^{-1}B_i'\mathcal{D}_{i+1}' \tag{40c}$$

where Θ_i^I, Ψ_i^I are as in (28). Note that here $\mathcal{D}_i$ describes a closed-loop transition matrix from $j\underline{N}+k$ to $i \geq (j-1)\underline{N}+k$ and $\mathcal{O}_i$ is some controllability Gramian, c.f. [11].

Then we compute $P_{j\underline{N}+k}$, $j = 1, \ldots, \eta-2$ by

$$P_{j\underline{N}+k} = P^I_{j\underline{N}+k} + \mathcal{D}'_{(j+1)\underline{N}+k}P^I_{(j+1)\underline{N}+k}\times (I + \mathcal{O}_{(j+1)\underline{N}+k}\Delta P_{(j+1)\underline{N}+k})^{-1}\mathcal{D}_{(j+1)\underline{N}+k} \tag{41a}$$

$$\Delta P_{(j+1)\underline{N}+k} = P_{(j+1)\underline{N}+k} - P^I_{(j+1)\underline{N}+k}. \tag{41b}$$

Finally, we can compute for $i = j\underline{N}-1+k, \ldots, (j-1)\underline{N}+k$, $j = 1, \ldots, \eta-1$ with $P_{j\underline{N}+k}$ from (41)

$$P_i = A_i'P_{i+1}A_i + \tilde{Q}_i - \Theta_i'\Psi_i^{-1}\Theta_i. \tag{42}$$

The computations (40) and (42) can be done for each part independently of the other parts, which allows parallel computations. In contrast the different parts are only coupled by (41). Note that we do not need to evaluate (40a), (40b), (42) for the last part ($j = \eta$) and (40a), (40b), (40c) for the first part ($j = 1$). Hence to balance the computational effort one can evaluate the first and last on the same computation unit. Also it is possible computation of the dual residual $\boldsymbol{\zeta}^d$ in such a way, c.f [11].

C. Memory demand

Let us discuss the memory demand of the proposed method. We need to store the problem setup, i.e., the data due to the dynamics (1), the constraints (5) and the cost (4). Moreover for the evaluation of the algorithm we need to store $\tilde{C}_{i,j}$, $\tilde{D}_{i,j}$ (23) and the data necessary for (27), see [12].

To simplify the presentation we assume in the remainder of this section $n_i = n$, $p_i = p$ except for $p_{k+N} = 0$. Moreover, we assume box constraints on the input and state, m^e additional quadratic constraints for each stage, for Q_i, R_i, V_i no special structure and $d_i = 0$, $q_i = 0$, $r_i = 0$.

In Table III we illustrate the memory demand assuming that symmetry is used to reduce the memory demand. We observe that the memory demand is linear in N.

In contrast if we would not exploiting the structure, then this would result in a memory demand of $O(N(Np^2+np)+(Nm^e+m^f)(N(n+Np)^2))$, which increases cubically with N or quadratically if $m^e = 0$, see [12] for details. This impressively underlines the possibility of decreased memory demand exploiting the structure.

D. Comparison with other methods

Let us shortly compare the proposed method with other methods assuming that there are no quadratic constraints: $m^e = 0$.

TABLE III

MEMORY DEMAND OF PROPOSED ALGORITHM

Data source	Memory demand
Dynamics (1) (A_i, B_i)	$Nn(n+p)$
Constraints (5) $(E_{i,j})$	$0.5Nm^e(n+p)^2$
Cost (4) (Q_i, R_i, V_i)	$0.5N(n+p)^2)$
Lin. constraints (23) $(C_{i,j}, D_{i,j})$	$Nm^e(n+p)$
Data for (27): solution via (28), (37) or (39)	$N(0.5p^2+np)$
Data for (27): solution via (38) (special case)	Nnp

TABLE IV

MEMORY DEMAND - COMPARISON WITH OTHER FORMULATIONS

Approach	Memory Demand
Solution with (28), (37) or (39)	$N(np+0.5p^2)$
Solution with (38) (special case)	Nnp
Naive Condensed	$0.5N^2p(p+m)+Nn(p+m)$
Rao et al. [21]	$0.5(N-1)n^2+0.5Np^2$
Wang & Boyd [24] (and so [6])	$(N-1)n^2+0.5Nn^2$
Mancuso & Kerrigan [17] $(n \geq p)$	$O(Nn(3n-p))$
Jerez et al. [10]	$O(Nn(p+n))$

1) Comparison of computational demand :
Without the quadratic constraints the computational effort of an interior point iteration of the proposed method is $O(N(n+p)^3)$, which is similar to [6], [21], [24]. The works [10], [17] have a computational effort of $O(Nn(3n-p)^2)$ and $O(N(n^3+n^2p))$, respectively, assuming for both $n \geq p$ and for [10] that the nilpotency index is as small as possible (which delivers the best performance). In all these works the computational effort grows linear with N. Without exploiting the structure in (8) the computational effort of each interior point iteration would be $O((Np)^3)$, which grows cubically with N and for each evaluation of (8) some $(O(Nnp))$ additional computations are needed for $\mathbf{h}(\hat{x}_k)$ and $G_{i,j}(\hat{x}_k)$.

2) Comparison of memory demand :
In Table IV we compare the memory demand of the proposed method with other interior point based approaches to solve (8). We observe that our approach has a memory demand linear in the number of states n, which is advantageous if there are many states and few inputs.

VI. SIMULATION EXAMPLE

As example we consider, a chain of $\rho \geq 4$ masses, where each mass is connected with their neighbors with springs and the outer masses also to walls. The left 4 masses have each an actuator, which can apply a force to it. This setup is illustrated in Figure 1. We assume the spring constants as 1, the inertias of the masses as 1 and that there is no damping.

We use a zero order hold and a sampling time of 0.5 to discretize the problem and choose the weighting matrices Q_i, R_i as identities except for Q_{k+N}, which is chosen from the LQR Riccati equation. The disturbance/exogenous input (d_i) and the other weightings (q_i, r_i, V_i) are set to zero.

The constraints are limitations of the position and speed of each mass to be ± 2 and of the actuator forces to ± 0.5.

To evaluate the algorithm we fixed the number of iterations to 10, and averaged the computation time over 1000 evaluations. We use a single core of a 3.4 GHz Intel CPU and compiled our C++ implementation of the algorithms with the GCC (G++) compiler, for more details see [12].

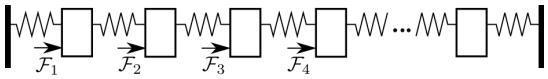

Fig. 1. Benchmark example: Masses connected by springs.

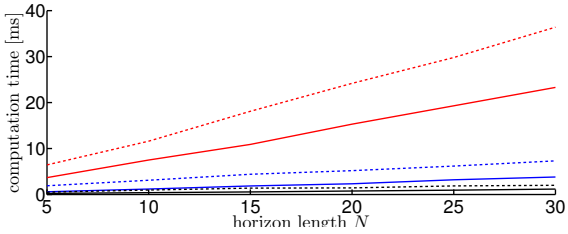

Fig. 2. Computation times for 10 interior point iterations. Black $\rho = 4$, blue $\rho = 10$, red $\rho = 20$. Solid: (38), (35a). Dashed: (39). Box constraints.

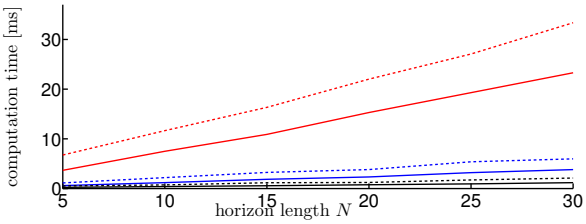

Fig. 3. Computation times using 10 interior point iterations. Black $\rho = 4$, blue $\rho = 10$, red $\rho = 20$. Solid: only box constraints. Dashed: setup with quadratic constraints.

First, we assume no additional quadratic constraints, i.e. (8) is a QP. In Figure 1 we show the computation times using the algorithm (38) (with (35a)) and the array algorithm (39) to solve the Riccati recursion for different number of masses and horizon length. The computational effort increases linearly with the horizon length, as expected. We observe that algorithm (38) is about twice as fast as the array algorithm (39). This might be due to two effect: (39) has a higher FLOP count than (35a), (38) and the matrix-matrix operations in (35a), (38) might be computed more efficiently than the QR decomposition in (39).

Second let us consider quadratic constraints. Therefore we replace the terminal box constraint by an ellipsoidal terminal constraint, [18], and we add so-called real-time constraints, [6], [28]. This setup results in an MPC problem with $n = 2\rho+1$ states and $p = \rho+1$ inputs and one quadratic constraint per stage (2 for the terminal state). Figure 2 shows the computation times for this setup and using only box-constraints as above. We use (28) and (38) repectively (both with (35)) to solve the Riccati recursion. The computational effort is by a factor of 1.45 (for the best case $N = 30$, $\rho = 20$) to 2.14 (for the worst case $N = 5$, $\rho = 4$) higher for the setup with quadratic constraints.

VII. Extension to Barrier method

Note that the above method to compute the Newton step can be also used for (approximate) primal barrier method. These methods have the advantage that they can easily be warm-started [24], in contrast to primal dual methods, c.f. [19], and one might guarantee certain control properties.

In feasible start, approximate primal barrier methods aim to solve the following optimization problem

$$\min_{\mathbf{u}} J^a(\mathbf{u};\hat{x}_k) = \min_{\mathbf{u}} J^c(\mathbf{u};\hat{x}_k) + \sum_{i=k}^{k+N}\sum_{j=1}^{m_i} \vartheta_{i,j}\Phi_{i,j}(\mathbf{u};\hat{x}_k) \tag{43}$$

where

$$\Phi_{i,j}(\mathbf{u};\hat{x}_k) = -\log(-\frac{1}{2}\|M_{i,j}\mathbf{u}\|_2^2 - G_{i,j}(\hat{x})\mathbf{u} + g_{i,j}(\hat{x})) \tag{44}$$

and $\vartheta_{i,j} > 0$ are called barrier parameters, see [19], [24].

Starting with a feasible $\mathbf{u}$ (43) is solved using a damped Newton method and fixed barrier parameters $\vartheta_{i,j}$.

For LTI systems (time-invariant dynamics and constraints) and quadratic costs one can guarantee for suitable designed terminal control laws, terminal sets and terminal conditions recursive feasibility and stability. Note that to guarantee stability one needs either additional constraints, c.f. [28] or in some cases one can tune the parameters $\vartheta_{i,j}$ such that $\min_{\mathbf{u}} J^c(\mathbf{u};\hat{x}_k)$ (plus/minus some constant) is a Lyapunov function, c.f. [12].

VIII. Summary and Outlook

This work presented an interior point method for predictive control of linear systems with quadratic constraints and a quadratic performance index. The proposed method has only the inputs as decision variables and exploits the problem structure. With respect to the implementation we discussed a deterministic line-search, solution methods for the Riccati recursion, special cases and the computational demand. As shown, the approach has partly significant advantages with respect to the computational and memory demand.

Further work includes additional comparisons with other methods and an extension to descriptor systems.

References

[1] Åkerblad, M., and Hansson, A. Efficient solution of second order cone program for model predictive control. *International Journal of Control 77*, 1 (2004), 55–77.

[2] Alessio, A., and Bemporad, A. A Survey on Explicit Model Predictive Control. In *Nonlinear Model Predictive Control*, Lecture Notes in Control and Information Sciences. Springer, 2009, pp. 345–369.

[3] Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. The explicit linear quadratic regulator for constrained systems. *Automatica 38*, 1 (2002), 3–20.

[4] Bemporad, A., and Patrinos, P. Simple and Certifiable Quadratic Programming Algorithms for Embedded Linear Model Predictive Control. In *Proc. 4th IFAC Nonlinear Model Predictive Control Conference* (2012), pp. 14 – 20.

[5] Bertsekas, D. P. *Nonlinear Programming*. Athena Scientific, 1999.

[6] Domahidi, A., Zgraggen, A., Zeilinger, M., Morari, M., and Jones, C. Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control. In *Proc. IEEE Conference on Decision and Control* (2012), pp. 668–674.

[7] Dunn, J., and Bertsekas, D. Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems. *Journal of Optimization Theory and Applications 63*, 1 (1989), 23–38.

[8] Ferreau, H. J., Bock, H. G., and Diehl, M. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control 18*, 8 (2008), 816–830.

[9] Gondhalekar, R., and Jones, C. N. MPC of constrained discrete-time linear periodic systems - A framework for asynchronous control: Strong feasibility, stability and optimality via periodic invariance. *Automatica 47*, 2 (2011), 326–333.

[10] Jerez, J., Kerrigan, E., and Constantinides, G. A sparse and condensed QP formulation for predictive control of LTI systems. *Automatica 48*, 5 (2012), 999–1002.

[11] Kailath, T., Sayed, A. H., and Hassibi, B. *Linear Estimation*. Prentice Hall, 2000.

[12] Kögel, M., and Findeisen, R. On structure-exploiting, condensed interior point methods for predictive control of linear system (Internal Report with Additional Material). Available on request.

[13] Kögel, M., and Findeisen, R. Fast predictive control of linear, time-invariant systems using an algorithm based on the fast gradient method and augmented Lagrange multipliers. In *Proc. IEEE Conference on Control Applications* (2011), pp. 780–785.

[14] Kögel, M., and Findeisen, R. Parallel solutions of model predictive control using the alternating direction method of multipliers. In *Proc. IFAC Conference on Nonlinear Model Predictive Control* (2012), pp. 369–374.

[15] Limon, D., Alvarado, I. A. T., and Camacho, E. MPC for tracking piecewise constant references for constrained linear systems. *Automatica 44*, 9 (2008), 2382–2387.

[16] Maciejowski, J. M. *Predictive control: With constraints*. Prentice Hall, 2002.

[17] Mancuso, G., and Kerrigan, E. Solving Constrained LQR Problems by Eliminating the Inputs from the QP. In *Proc. 50th IEEE Conference on Decision and Control and European Control Conference* (2011), pp. 507–512.

[18] Mayne, D., Rawlings, J., Rao, C., and Scokaert P.O.M. Constrained model predictive control: Stability and optimality. *Automatica 36*, 6 (2000), 789–814.

[19] Nocedal, J., and Wright, S. J. *Numerical Optimization*. Springer, 2000.

[20] Qin, S.J. , and Badgwell, T.A. A survey of industrial model predictive control technology. *Control Engineering Practice 11*, 7 (2003), 733–764.

[21] Rao, C., Wright, S., and Rawlings, J. Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications 99*, 3 (1998), 723–757.

[22] Richter, S., Morari, M., and Jones, C. N. Towards Computational Complexity Certification for Constrained MPC Based on Lagrange Relaxation and the Fast Gradient Method. In *Proc. 50th IEEE Conference on Decision and Control and European Control Conference* (2011), pp. 5223 – 5229.

[23] Shahzad, A., Kerrigan, E., and Constantinides, G. A fast well-conditioned interior point method for predictive control. In *Proc. 49th IEEE Conference on Decision* (2010), pp. 508–513.

[24] Wang, Y., and Boyd, S. Fast Model Predictive Control Using Online Optimization. *IEEE Transactions on Control Systems Technology 18*, 2 (2010), 267–278.

[25] Wright, S. J. *Primal-Dual Interior-Point Methods*. SIAM, 1997.

[26] Zavala, V. M., and Biegler, L. T. Nonlinear programming strategies for state estimation and model predictive control. In *Nonlinear model predictive control*. Springer-Verlag, 2009, pp. 419–432.

[27] Zavala, V. M., Laird, C. D., and Biegler, L. T. Fast implementations and rigorous models: Can both be accommodated in NMPC? *International Journal of Robust and Nonlinear Control 18*, 8 (2008), 800–815.

[28] Zeilinger, M., Jones, C., Raimondo, D., and Morari, M. Real-time MPC-Stability through robust MPC design. In *Proc. 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference* (2009), pp. 3980–3986.