# On Tailored Model Predictive Control for Low Cost Embedded Systems with Memory and Computational Power Constraints

Markus Kögel and Pablo Zometa and Rolf Findeisen

***Abstract*— Even though many efficient formulations and implementations exist by now, predictive control on low cost embedded systems with constrained memory and computing power is still challenging. We present an algorithm combining Nesterov's gradient method and the method of multipliers for linear model predictive control, which can exploit the structure and does not need slack variables. Moreover, we discuss implementation issues focusing on embedded systems. We examine the performance using a benchmark example and illustrate the suitability for embedded system using a simple mechatronic system with state and input constraints and a low-cost microcontroller.**

## I. Introduction

Model predictive control (MPC) enables control of constrained systems with a high performance, see e.g. [6], [10], [17]. MPC uses as feedback the solution of an optimal control problem: at each sampling instance, an optimal control problem is solved and the first part of the resulting optimal input is applied until the next sampling instance.

Unfortunately, the computational demand of MPC is often a challenge, especially for embedded hardware with limited computational power. Therefore the efficient and tailored solution of the underlying optimization problem is of interest, because it allows to use cheaper hardware, to treat more complex problems or to increase the sampling rate.

Here we present and evaluate an efficient method for MPC of linear systems with polytopic state and input constraints and a quadratic cost criterion. In detail, we propose a method using online-optimization, i.e., the optimal control problem, a quadratic program, is solved online at each step.

In contrast, in explicit MPC one determines offline control laws for all possible states. At each sampling instance one needs to find and evaluate the correct control law, see [1]. However the memory demand of explicit MPC grows in general exponentially in the number of states, inputs and horizon, thus it is usually restricted to small-scale systems.

By now different online-optimization approaches exist. Often interior point methods are considered for MPC. For example the works [11], [12], [18], [22] present tailored methods, which utilize the inherent structure of MPC problems. Also predictive control based on active set methods has been investigated, we refer to the works [5], [14]. In [19] Nesterov's gradient method is considered for predictive control of systems with input constraints. In [20] it is combined with partial Lagrange relaxation to handle also state constraints.

For input constrained MPC using Nesterov's method we outlined in [7] a method to determine the gradient exploiting the problem structure and in [24] an implementation on an embedded system. In [8], [9] we presented different combinations of Nesterov's method with the method of multipliers to consider in addition state constraints.

Here we present an algorithm also based on Nesterov's gradient method and the method of multipliers, which solves as [9] the condensed quadratic program. However in contrast to [9], this algorithm uses one multiplier per two-sided constraint and does not use slack variables, which enables a faster solution. Although the objective function of the subproblem is not everywhere twice differentiable, we show how the arising subproblem can be solved using Nesterov's gradient method. Additionally, the algorithm can utilize the problem structure, which results in a memory demand and time complexity linear in the horizon length. We illustrate the efficiency of the proposed method via an implementation on an embedded system and a benchmark example.

The remainder of the paper is structured as follows. First we discuss the problem setup in Section II. In Section III we apply the method of multipliers to the MPC problem. Afterwards in Section IV, we analyze the subproblem arising from the multiplier method and its solution using Nesterov's method. In Section V we discuss implementation issues. In Section VI examples illustrate the performance and applicability for embedded control of the proposed algorithm.

The notation is mainly standard. All norms are Euclidean norms. $\langle x, y\rangle$ is the inner product. $\lambda_{Min}(M)$ denotes the smallest eigenvalue of a symmetric matrix $M$. For a vector $v$, $v_i$ is its $i$th entry. $M_i$ is the $i$th row of the matrix $M$.

## II. Model predictive control setup

In this work we consider model predictive control of a linear, time-invariant, discrete-time plant subject to constraints on the states and input and a quadratic cost criterion.

In detail, the linear plant is given by

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k), \tag{1}$$

where $x(t_k) \in \mathbb{R}^n$ is the state and $u(t_k) \in \mathbb{R}^p$ the input.

Additionally, we assume that the constraints are split into two classes. First the input $u(t_k)$ needs to satisfy box constraint

$$\underline{u} \leq u(t_k) \leq \overline{u}. \tag{2}$$

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

M. K. is supported in part by the International Max Planck Research School Magdeburg, Germany.

Moreover, there are additional constraints on state and input

$$\underline{e} \leq Cx(t_k) + Du(t_k) \leq \overline{e}, \tag{3}$$

where $C \in \mathbb{R}^{q \times n}$ and $D \in \mathbb{R}^{q \times p}$. This classification simplifies later the formulation of the algorithm.

Since we use a control and prediction horizon of $N$, let us introduce the input sequence $\mathbf{u}$ and state trajectory $\mathbf{x}$

$$\mathbf{x} = \begin{pmatrix} x(k)^T, & \dots & x(k+N)^T \end{pmatrix}^T \tag{4a}$$

$$\mathbf{u} = \begin{pmatrix} u(k)^T, & \dots & u(k+N-1)^T \end{pmatrix}^T, \tag{4b}$$

as optimization variables[1], where $\mathbf{u} \in \mathbb{R}^{Np}$ and $\mathbf{x} \in \mathbb{R}^{(N+1)n}$. Clearly, $\mathbf{u}$ and $\mathbf{x}$ need to be consistent with the plant dynamics (1) and the current state $x(t_k)$. In particular, we need to have

$$x(k) = x(t_k). \tag{5}$$

Additionally, $x(i)$ and $u(i)$ need to satisfy the constraints (2), (3) for $i = k, \dots, k+N-1$. The terminal state $x(k+N)$ need to satisfy a terminal constraint given by

$$\underline{f} \leq Fx(k+N) \leq \overline{f}, \tag{6}$$

where $F \in \mathbb{R}^{r \times n}$.

The considered convex quadratic cost criterion $J$ is

$$J = \sum_{i=0}^{N-1} J^s(x(i+k), u(i+k)) + J^f, \tag{7}$$

where $J^s$ is the stage cost and $J^f$ is the terminal cost

$$J^s(x(j), u(j)) = \frac{1}{2} \begin{pmatrix} x(j) \\ u(j) \end{pmatrix}^T W \begin{pmatrix} x(j) \\ u(j) \end{pmatrix} \tag{8a}$$

$$W = W^T = \begin{pmatrix} Q & S^T \\ S & R \end{pmatrix} \geq 0 \tag{8b}$$

$$J^f = \frac{1}{2} x(N+k)^T T x(N+k),\ T \geq 0. \tag{8c}$$

For possible choices of terminal constraints and costs we refer to [13] and the references therein.

In summary, we need to determine $\mathbf{u}$ and $\mathbf{x}$, which minimizes the cost function (7), is consistent with the plant dynamic (1) and the current state (5) and satisfies the inequality constraints (2), (3), (6).

In this work we use a condensed formulation, which means that the equality constraints (1) and (5) are used to eliminate $\mathbf{x}$. The resulting quadratic program (QP) has only $\mathbf{u}$ as optimization variable and is given by

$$\min_{\underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}} \left( \frac{1}{2} \mathbf{u}^T H \mathbf{u} + \mathbf{u}^T g(x(t_k)) \right) \tag{9a}$$

$$\text{s.t. } \underline{\mathbf{z}}(x(t_k)) \leq E\mathbf{u} \leq \overline{\mathbf{z}}(x(t_k)), \tag{9b}$$

where $\underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}$ denotes the set given by

$$\underline{u} \leq u(i) \leq \overline{u},\ i = k, \dots, k+N-1. \tag{10}$$

The matrix $E$ and vectors $\overline{\mathbf{z}}(x(t_k)), \underline{\mathbf{z}}(x(t_k))$ depend on the constraints (3), (6) and the dynamic (1) and are given in Appendix A. The cost terms $g(x(t_k))$, $H$ result from the system dynamic (1) and the cost criterion (7), see e.g. [7].

In summary, at each sampling instance $t_k$ the optimization problem (9), which depends on $x(t_k)$, is solved to obtain the feedback $u(t_k) = \hat{u}(k)$ as first part of the optimal input $\hat{\mathbf{u}}$.

In the next sections we outline and evaluate a method to solve (9) using a combination of the method of multipliers and Nesterov's gradient method.

## III. Method of multipliers

In this section we apply the *method of multipliers*, also called *(partial) augmented Lagrangian method*, to the MPC problem (9). In detail, we use the method of multipliers to treat the inequality constraints (3), (6), see e.g. [2, Ch. 3, Ch. 5] for more details.

Introducing slack variables $\mathbf{s}$ an equivalent formulation of the optimization problem (9) is given by

$$\min_{\mathbf{s}, \underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}} \left( \frac{1}{2} \mathbf{u}^T H \mathbf{u} + \mathbf{u}^T g(x(t_k)) \right) \tag{11a}$$

$$\text{subject to } \underline{\mathbf{z}}(x(t_k)) \leq E\mathbf{u} + \mathbf{s} \leq \overline{\mathbf{z}}(x(t_k)),\ \mathbf{s} = 0. \tag{11b}$$

Applying the method of multipliers to the equality $\mathbf{s} = 0$, see [2], yields the subproblem

$$\min_{\mathbf{s}, \underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}} \left( \frac{1}{2} \mathbf{u}^T H \mathbf{u} + \mathbf{u}^T g(x(t_k)) + \xi^T \mathbf{s} + \frac{c}{2} \|\mathbf{s}\|_2^2 \right) \tag{12a}$$

$$\text{subject to } \underline{\mathbf{z}}(x(t_k)) \leq E\mathbf{u} + \mathbf{s} \leq \overline{\mathbf{z}}(x(t_k)), \tag{12b}$$

where $\xi$ is the so-called multiplier and $c, c > 0$ is the so-called penalty parameter, which can be freely chosen.

The method of multipliers is iterative and each iteration consists of two steps. First, we solve (12) using a multiplier $\xi^i$ and obtain the minimizers $\hat{\mathbf{s}}^i$ and $\hat{\mathbf{u}}^i$. Afterwards, we determine a new multiplier $\xi^{i+1}$ using the multiplier update

$$\xi^{i+1} = \xi^i + c\hat{\mathbf{s}}^i. \tag{13}$$

The subproblem (12) features the slack variables $\mathbf{s}$. One can eliminate these slack variables $\mathbf{s}$ from the subproblem (12) by first minimizing with respect to them, see [2]. In our case this yields the subproblem

$$\min_{\underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}} f^a(\mathbf{u}; \xi, x(t_k)), \tag{14}$$

with the objective function $f^a$ parameterized by $\xi$ and $x(t_k)$

$$f^a(\mathbf{u}; \xi, x(t_k)) = \frac{1}{2} \mathbf{u}^T H \mathbf{u} + \mathbf{u}^T g(x(t_k)) + P(\mathbf{u}; \xi, x(t_k)). \tag{15}$$

Here the function $P(\mathbf{u}; \xi, x(t_k))$ is given by

$$P(\mathbf{u}; \xi, x(t_k)) = \min_{\mathbf{s}} \left( \xi^T \mathbf{s} + \frac{c}{2} \|\mathbf{s}\|_2^2 \right) \tag{16}$$

$$\text{subject to } \underline{\mathbf{z}}(x(t_k)) \leq E\mathbf{u} + \mathbf{s} \leq \overline{\mathbf{z}}(x(t_k)).$$

Since $P(\mathbf{u}; \xi, x(t_k))$ is a sum of scalar optimization problems, we can solve it analytically, compare [2]. In particular,

[1]For the optimization variables we use the notation $x(k)$, $u(k)$ to distinguish them from the actual state $x(t_k)$ and input $u(t_k)$.

the optimal $\hat{\mathbf{s}}(\mathbf{u};\xi,x(t_k))$ is given by

$$\hat{\mathbf{s}}_j(\mathbf{u};\xi_j,x(t_k)) = \begin{cases} E_j\mathbf{u}-\overline{\mathbf{z}}_j(x(t_k)), & \text{if } \mathcal{S}_j^+(\mathbf{u};\xi_j,x(t_k))>0 \\ E_j\mathbf{u}-\underline{\mathbf{z}}_j(x(t_k)), & \text{if } \mathcal{S}_j^-(\mathbf{u};\xi_j,x(t_k))<0 \\ -\frac{1}{c}\xi_j, & \text{else.} \end{cases} \quad (17)$$

Here we use as shorthand

$$\mathcal{S}_j^+(\mathbf{u};\xi_j,x(t_k)) = \xi_j + c(E_j\mathbf{u}-\overline{\mathbf{z}}_j(x(t_k))) \quad (18a)$$

$$\mathcal{S}_j^-(\mathbf{u};\xi_j,x(t_k)) = \xi_j + c(E_j\mathbf{u}-\underline{\mathbf{z}}_j(x(t_k))). \quad (18b)$$

So, $P(\mathbf{u};\xi,x(t_k))$ (16) is given analytically by

$$P(\mathbf{u};\xi,x(t_k)) = \sum_{j=1}^{Nq+r} P_j(\mathbf{u};\xi_j,x(t_k)) \quad (19a)$$

$$P_j(\mathbf{u};\xi_j,x(t_k)) = \xi_j\hat{\mathbf{s}}_j(\mathbf{u};\xi_j,x(t_k)) + \frac{c}{2}\hat{\mathbf{s}}_j(\mathbf{u};\xi_j,x(t_k))^2, \quad (19b)$$

where $\hat{\mathbf{s}}_j(\mathbf{u};\xi_j,x(t_k))$ is as in (17).

We refer to the next section for a discussion of properties of the subproblem (14) and its solution based on Nesterov's gradient method.

Using the minimizer $\hat{\mathbf{u}}^i$ of (14) we can rewrite the multiplier update (13) using (17) and (18) as

$$\xi_j^{i+1} = \begin{cases} \mathcal{S}_j^+(\hat{\mathbf{u}}^i;\xi_j^i,x(t_k)), & \text{if } \mathcal{S}_j^+(\hat{\mathbf{u}}^i;\xi_j^i,x(t_k))>0 \\ \mathcal{S}_j^-(\hat{\mathbf{u}}^i;\xi_j^i,x(t_k)), & \text{if } \mathcal{S}_j^-(\hat{\mathbf{u}}^i;\xi_j^i,x(t_k))<0 \\ 0, & \text{else.} \end{cases} \quad (20)$$

In summary, the presented multiplier method solves the problem (9) by solving the subproblem (14) and updating the multipliers (20) as outlined in Algorithm 1.

**Algorithm 1** Multiplier method

**Require:** Initial guess $\xi^0 \geq 0$, state $x(t_k)$, number of iterations $i_{max}$

1: **for** $i = 1,\ldots,i_{max}$ **do**
2: Solve subproblem (14) and obtain $\hat{\mathbf{u}}^i$
3: Compute $\xi^i$ using multiplier update (20)
4: **end for**
5: **return** $\xi^{i_{max}}$, $\hat{\mathbf{u}}^{i_{max}}$

Note that the elimination of the slack variables is basically only an approach to solve the subproblem arising from the multiplier method. Thus one could use in Algorithm 1 also (12) and (13), instead of (14) and (20), respectively.

*Remark 1:* (Convergence and inexact minimization) Since the function $\frac{1}{2}\mathbf{u}^T H\mathbf{u} + \mathbf{u}^T g(x(t_k))$ is convex, (10) is closed and convex and the constraints (3), (6) are linear in $\mathbf{u}$ any value of $c>0$ guarantees convergence of the multiplier method [2], [4], [21] (assuming a precise enough solution of (14)). In general larger values of $c$ improve the convergence, but lead to a more difficult subproblem as outlined in the next section.

In this work we focus on the application of this multiplier method to the MPC problem (9) and implementation issues, e.g., the solution of the subproblem (14) using Nesterov's method. Determining or discussing analytical convergence bounds of the multiplier method or the influence of inexact minimization of (14) is beyond the scope of this work.

*Remark 2:* (Different update schemes) There exist multiplier update schemes different from (20). For example it is possible to use Nesterov's gradient method for the update or a second order update, compare [2], [3]. Unfortunately, using Nesterov's gradient method for the multiplier update requires a precise solution of subproblem (14), due to its inherent error accumulation, see [3]. Also a second order update requires a precise solution and has also a larger computation effort than (20), compare [2]. Thus, we use the simple update (20), which seems to be more robust to inexact solution of subproblem (14), compare [2], [4], [21].

## IV. Solution and properties of the subproblem

In this section we outline a solution of subproblem (14) using Nesterov's gradient method, also known as *Fast Gradient* method, [15], [16]. We first present this method and then investigate properties of the subproblem.

### A. Nesterov's gradient method

We use Nesterov's gradient method [15], [16] to solve (14), which requires that $f^a$ is convex and has a Lipschitz continuous gradient. So there need to be for all $\mathbf{u}\in\mathbb{R}^{Np}$, $\mathbf{v}\in\mathbb{R}^{Np}$ constants $L$ and $\phi\geq 0$ such that

$$\|\nabla f^a(\mathbf{u};\xi,x(t_k)) - \nabla f^a(\mathbf{v};\xi,x(t_k))\| \leq L\|\mathbf{u}-\mathbf{v}\|, \quad (21)$$

$$\langle \nabla f^a(\mathbf{u};\xi,x(t_k)) - \nabla f^a(\mathbf{v};\xi,x(t_k),\mathbf{u}-\mathbf{v}\rangle \geq \phi\|\mathbf{u}-\mathbf{v}\|^2. \quad (22)$$

$L$ is a Lipschitz constant of the gradient and $\phi$ is called strong convexity parameter. In Proposition 1 we show that there are indeed such constants and how to compute them.

Nesterov's gradient method is presented in Algorithm 2. The method uses the projected gradient step

$$\mathcal{G}_{\mathcal{U}}(\mathbf{u};\xi,x(t_k)) = \mathcal{P}_{\mathcal{U}}\left(\mathbf{u} - \frac{1}{L}\nabla f^a(\mathbf{u};\xi,x(t_k))\right), \quad (23)$$

where $\mathcal{P}_{\mathcal{U}}$ denotes the Euclidean projection onto the set $\mathcal{U}$. In our case we have $\mathcal{U} = \{\mathbf{u} \text{ s.t. } \underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}\}$, compare (10). The projection onto this box set is the saturation

$$\mathcal{P}_{\mathcal{U}}(\mathbf{u}) = \max(\underline{\mathbf{u}}, \min(\overline{\mathbf{u}}, \mathbf{u})), \quad (24)$$

where $\max$ and $\min$ are applied entrywise.

The scalar sequence $\gamma^i$ (step 4 of Algorithm 2) needs to satisfy certain conditions and depends on the strong convexity parameter $\phi$, see [16]. In particular, if $\phi>0$, then one can choose $\gamma^i$ constant: $\gamma^i = \frac{\sqrt{L}-\sqrt{\phi}}{\sqrt{L}+\sqrt{\phi}}$.

*Remark 3:* (Other input constraint sets $\mathcal{U}$) One can use other closed and convex set $\mathcal{U}$ instead of the box set $\underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}$. In general one need to solve an optimization problem for the Euclidean projection $\mathcal{P}_{\mathcal{U}}$. In practice this limits $\mathcal{U}$ to sets for which the projection is efficiently possible, see e.g. [16], [19].

**Algorithm 2** Nesterov's gradient method

**Require:** Initial guess $\mathbf{u}^0$, number of iterations $i_{max}^{NGM}$, Lipschitz constant $L$ and sequence $\gamma^i$

Set $\mathbf{v}^0 = \mathbf{u}^0$
**for** $i = 1, \dots, i_{max}^{NGM}$ **do**
  Compute $\mathbf{u}^i = \mathcal{G}_{\mathcal{U}}(\mathbf{v}^{i-1}; \xi, x(t_k))$
  Compute $\mathbf{v}^i = \mathbf{u}^i + \gamma^i(\mathbf{u}^i - \mathbf{u}^{i-1})$
**end for**
**return** $\mathbf{u}^{i_{max}^{NGM}}$

### B. Properties of the subproblem

In this part we investigate the properties of the subproblem (9) and discuss how to compute the constants $L$ and $\phi$ necessary for Nesterov's gradient method.

Computing the gradient of $f^a(\mathbf{u}; \circ)$[2] (15) using (17) yields

$$\nabla f^a(\mathbf{u}; \circ) = H\mathbf{u} + g(x(t_k)) + \sum_{j=1}^{Nq+r} \nabla P_j(\mathbf{u}; \circ_j) \tag{25}$$

$$\nabla P_j(\mathbf{u}; \circ_j) \tag{26}$$
$$= \begin{cases} E_j^T(cE_j\mathbf{u} - c\overline{\mathbf{z}}_j(x(t_k)) + \xi_j), & \text{if } \mathcal{S}_j^+(\mathbf{u}; \circ_j) > 0 \\ E_j^T(cE_j\mathbf{u} - c\underline{\mathbf{z}}_j(x(t_k)) + \xi_j), & \text{if } \mathcal{S}_j^-(\mathbf{u}; \circ_j) < 0 \\ 0, & \text{else.} \end{cases}$$

We observe that the gradient exists everywhere. Note that we can compute $\nabla P(\mathbf{u}; \circ)$ using

$$\nabla P(\mathbf{u}; \circ) = E^T \left(\max(\mathcal{S}^+(\mathbf{u}; \circ), ) + \min\left(\mathcal{S}^-(\mathbf{u}; \circ), 0\right)\right), \tag{27}$$

compare (18) and (25).

In contrast the Hessian $\mathcal{H}(f^a(\mathbf{u}; \circ))$ exists, if and only if,

$$\mathcal{S}_j^+(\mathbf{u}; \circ_j) \neq 0 \qquad \mathcal{S}_j^-(\mathbf{u}; \circ_j) \neq 0$$

for all $j = 1, \dots, Nq + r$ with $E_j \neq 0$ due to the switching in (26). Hence, $f^a$ is in general not everywhere twice differentiable. If the Hessian exists, it is given by

$$\mathcal{H}(f^a(\mathbf{u}; \circ)) = H + \sum_{j=1}^{Nq+r} \mathcal{H}(P_j(\mathbf{u}; \circ_j)))$$

$$\mathcal{H}(P_j(\mathbf{u}; \circ_j) = \begin{cases} cE_j^T E_j, & \text{if } \mathcal{S}_j^+(\mathbf{u}; \circ_j)) > 0 \\ & \text{or } \mathcal{S}_j^-(\mathbf{u}; \circ_j)) < 0 \\ 0, & \text{else.} \end{cases}$$

Although $f^a$ is not everywhere twice differentiable, we can determine constants $L$ and $\phi$ for Nesterov's method using the following proposition.

*Proposition 1: (Lipschitz and strong convexity constant) A strong convexity constant of $f^a$ is $\phi = \lambda_{Min}(H)$ and $L = \|H + cE^T E\|$ is a Lipschitz constant of $\nabla f^a$.*
The proof is given in Appendix C.

Note that determining $L$ and $\phi$ is straightforward and can be done offline.

The worst case convergence of Nesterov's gradient method depends on the condition number $\kappa = L\phi^{-1} \geq 1$, compare [16]. The next proposition describes the asymptotic influence of the penalty parameter $c$ on the condition number $\kappa$ .

*Proposition 2: (Influence of $c$ on $\kappa$)*
*a) $L$ is bounded above and below by an increasing function, which is affine in $c$.*
*b) If $c \to 0$, then $\kappa \to \frac{\|H\|}{\phi}$.*
*c) If $c \to \infty$, then $\kappa \to \infty$.*
The proof is provided in Appendix D.

So for very large penalty parameters $c$ subproblem (14) might be more difficult to solve whereas for small $c$ the condition number $\kappa$ is only slightly larger than the condition number of $H$. Thus for the choice of $c$ a trade-off is necessary: choosing a large $c$ requires to spend more computational effort on the solution of the subproblem, but can improve the convergence rate of the multiplier method.

## V. Implementation

In this section we discuss implementation issues such as structure exploitation, warm-starting and the computational demand. We also compare the proposed algorithm with our previous work [9] and outline the improvements.

### A. Computational demand and use of problem structure

There are two methods to compute the gradient $\nabla f^a$. First one can directly evaluate (27) and $H\mathbf{u} + g(x(t_k))$ or one can use the problem structure, similarly as in [7], [9].

Direct evaluation of (27) and $H\mathbf{u} + g(x(t_k))$ requires per iteration three matrix-vector multiplications, with $E$, $E^T$ and $H$, respectively[3]. So the overall time complexity is quadratic in $N$ due to the size of $E$ and $H$. In addition, we need to compute $g(x(t_k))$ and $\overline{\mathbf{z}}(x(t_k))$, $\underline{\mathbf{z}}(x(t_k))$ (35) once per sampling instance, which results in the time complexity reported in Table I.

Using the problem structure to compute the gradient $\nabla f^a$, yields the computational demand presented in Table I, which is only linear in $N$, compare [7], [9].

Finally, let us outline the memory demand of the proposed MPC algorithm, where we distinguish between dynamic data (variables) and static data (constants). Table I illustrates the memory demand. Note that for simplicity we neglect the data that does not depend on $N$.

In summary, using the structure provides an advantage for problems where the horizon is large compared to the number of states, since for this approach the memory demand and the time complexity per iteration are only linear in $N$.

### B. Choice of initial guesses / Warm-starting

We need initial guesses for Algorithms 1 and 2.

If we restart the overall algorithm, then we can use either an initial guess based on the previous solution (warm-starting) or zero as initial guess (cold-starting). Warm-starting delivers usually better results, in particular for accurate models and if the disturbances are small.

Additionally, if we need to solve a subproblem for different multipliers $\xi$, but with the same $x(t_k)$, i.e., within Algorithm 1, then we can also use warm-starting.

[2] We use the shorthand $\circ = (\xi, x(t_k))$ and $\circ_j = (\xi_j, x(t_k))$

[3] We need to compute only $E\mathbf{u}$ once for $\mathcal{S}^+$ and $\mathcal{S}^-$.

| | Direct method | Structure exploiting method |
|---|---|---|
| Overall time complexity | $O(Np+Nq+r+n)(n+p)) + O(\Psi(Np)^2 + \Psi Np(Nq+r))$ | $O(\Psi(Np+Nq+r+n)(n+p))$ |
| Demand of Static Data | $(Np)^2 + Np(Nq+r)$ | $2Np(n+p)$ |
| Demand of Dynamic Data | $3(Nq+r)+4Np$ | $2(Nq+r)+4Np$ |

TABLE I

OVERALL TIME COMPLEXITY AND MEMORY DEMAND FOR DIRECT AND STRUCTURE EXPLOITING METHOD ($\Psi = i_{Max}^{ALM} i_{Max}^{NGM}$).

### C. Further implementation issues

We want to mention that the algorithm does not require divisions, if one computes $\frac{1}{L}$ offline, and no square-roots. This is advantageous for embedded hardware, which does divisions and square-roots a lot slower than additions and multiplications. Moreover, one can avoid the multiplication with $\frac{1}{L}$ in (23) by scaling the data offline appropriately.

Note that the proposed algorithm is rather simple to implement: it requires only simple linear algebra such as matrix-vector multiplications, vector additions and comparisons.

### D. One sided constraints

We can consider also one sided constraints

$$C^{os}x(k) + D^{os}u(k) \leq \overline{e}^{os} \tag{29}$$

by considering them as two sided constraints using (3)

$$\begin{pmatrix} \underline{e} \\ \mu \end{pmatrix} \leq \begin{pmatrix} Cx(k)+Du(k) \\ C^{os}x(k)+D^{os}u(k) \end{pmatrix} \leq \begin{pmatrix} \overline{e} \\ \overline{e}^{os} \end{pmatrix}, \tag{30}$$

where $\mu$ is small enough. By adapting (19) and (20) we can actually implement $\mu = -\infty$ for the one sided constraints: $\mathcal{S}_j^- > 0$ will always be satisfied for this $\mu$, so we need evaluate only $\mathcal{S}_j^+$ for these constraints.

### E. Comparison with previous work [9]

Note that in [9] we used a different approach with slack variables and two multiplier per two-sided constraints.

In the case that all constraints are two-sided the condition number $\tilde{\kappa}$ of the subproblem in [9] satisfies

$$\tilde{\kappa} \geq \frac{\|H + 2cE^TE\|}{\min(c, \lambda_{Min}(H))} \geq \kappa = \frac{\|H + cE^TE\|}{\lambda_{Min}(H)},$$

compare Proposition 1 and [9]. Hence (14) has a lower condition number than the subproblem in [9], which enables in general a faster solution.

In addition, the proposed algorithm has a slightly smaller memory demand and computational effort: the computational demand per iteration of [9] is $O(Nq+r)$ larger and the amount of dynamic data is $8(Nq+r)$ larger.

## VI. EXAMPLES

We illustrate the proposed algorithm using a benchmark example and an implementation on an embedded system.

### A. Embedded system implementation

We present now an implementation of the proposed algorithm using a low-cost microcontroller unit (MCU). We briefly explain the tuning of the algorithm and discuss the results.

*1) System description:* We consider a two-link robotic arm build with the Lego NXT platform (Fig. 1). Our goal is to use MPC to drive the system to the origin with limited speed starting from an arbitrary position in the horizontal plane (gravitation does not influence the system). Link $i = 1, 2$ is described by states $\theta_i$ and $\omega_i$. $\theta_i$ is the angular position measured by an incremental encoder. $\omega_i$ is the angular speed, which is estimated by discrete differentiation of $\theta_i$ plus a low-pass filter. Link $i$ is actuated by a motor via pulse-width-modulated (PWM) voltage $u_i$ in percentage. Link 1 is constrained in the input and states as $-100 \leq u_1 \leq 100$, and $-1 \leq \omega_1 \leq 1$. Link 2 is constrained as $-25 \leq u_2 \leq 25$, and $-1 \leq \omega_2 \leq 1$. See Appendix B for the continuous-time system matrices. We discretized our system using a zero-order hold discretization.

The NXT electronics include an ARM7TDMI processor core running at 48 MHz, and 64 kB of memory. It lacks a floating-point unit (FPU), which makes floating-point arithmetic around 4 times slower than fixed-point arithmetic. For an in-depth description of the test platform see [24].

*2) Algorithm tuning:* The design of an MPC controller allows some freedom in the choice of application dependent parameters, like the sampling time and horizon length. To simplify the discussion, we take them as fixed to the following values: sampling time of 4 ms, horizon length of 5 steps. In an initial test, we determined the computational capabilities of our hardware for the given setup. We could only perform in real-time 1 iteration of Algorithm 2 (the main computational burden) using single-precision floating-point arithmetic. The use of $Q13.18$ fixed-point arithmetic[4] instead yields up to 6 iterations but with an inferior numeric precision. This low number of iterations requires a high penalization of the constraint violation, i.e. a large $c$, and a low condition number $\kappa$ to obtain an acceptable approximation of the QP solution.

Note that we have $\kappa \leq \|H\|\phi^{-1} + c\|E^TE\|\phi^{-1}$ from Proposition 1. To get a good trade-off between $c$ and $\kappa$, we tune the weighting matrices $Q$, $R$, $S$, and $T$ to obtain a good control performance of the MPC controller and at the same time a low $cond(H) = \|H\|\phi^{-1}$, similar as in [23]. Afterwards, we increase $c$ as long as $\kappa$ is acceptable. For this example using the weighting matrices reported in Appendix B and $c = 2000$ we have $\kappa = 10$.

*3) Results and Discussion:* Fig. 2 shows the step response of our system for an initial conditions $x(t_0)^T = [2.6\ 0.0\ 3.5\ 0.0]$, where we only plot the behavior of link

[4] $Q13.18$ denotes 32-bit binary fixed-point numeric representation, with 13 integer bits plus 1 sign bit and 18 fractional bits.

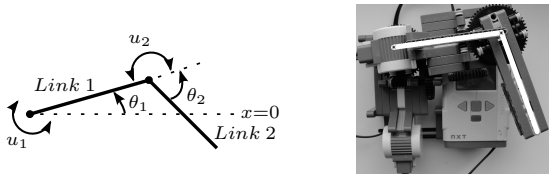


Fig. 1. The 2-link robotic arm used as test platform.

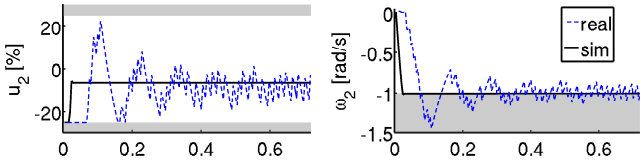


Fig. 2. Step response for link 2. The constraints are shown in grey. Continuous lines for simulated plant. Dashed lines for real plant.

2, as link 1 showed a similar trend. We perform 3 iterations for (20) and 2 for subproblem (14), with warm-starting and $Q13.18$ for both plots (simulated and real). The computation time on the MCU is 3.5 ms and the compiled code size is around 15 kB. Using the nominal plant, the MPC on the MCU is able to satisfy the input and state constraint (continuous line in Fig. 2). The difference to exact solution (not shown in the figure) is negligible for practical purposes. In other words without model plant mismatch and state estimation errors the presented algorithm is applicable on the MCU.

We repeat under the same conditions using the actual 2-link arm (dashed line in Fig. 2). We see that the input constraints are always satisfied. The constraint on $\omega_2$ is violated by around 40% at the beginning of the motion. This is due to model plant mismatch and state estimation errors. After 300 ms the MPC controller is able to satisfy the constraint on $\omega_2$ with violations of less than 15% [5].

Note that this problem can be split into two separated MPC controllers (the two links are not coupled), which would allow to solve the problem about twice as fast. Our intention with this example is, however, to show that similar problems can be solved with low-cost embedded microcontrollers, with limited computational power, low numeric precision, and little memory, using the ideas presented in this paper.

### B. Benchmark example

To evaluate the performance for larger systems and compare it with other solution methods we present another example. We consider six masses connected by springs to each other and to walls, as illustrated in Figure 3. Actuators can apply a force to each mass. We assume that the actuators have a maximum force and a limited slew rate.

In detail, we use the following continuous time model

$$\ddot{\rho}_i(t) = -2\rho_i(t) + \rho_{i+1}(t) + \rho_{i-1}(t) + \mathcal{F}_i(t), \qquad (31)$$

where $\rho_1, \ldots, \rho_6$ are the position of the masses, the actuator forces are $\mathcal{F}_1, \ldots, \mathcal{F}_6$ and $\rho_0 = \rho_7 = 0$ (solid walls). Note that this system is stable, but not asymptotically stable, since there is no damping.

For the overall model we consider the actuator force $\mathcal{F}$ as state and the rate $\Delta\mathcal{F}$ as input. This results in

$$x(t_{k+1}) = \begin{pmatrix} A^{cm} & B^{cm} \\ 0^{6\times 6} & I^{6\times 6} \end{pmatrix} x(t_k) + \begin{pmatrix} 0^{6\times 6} \\ I^{6\times 6} \end{pmatrix} u(t_k),$$

where the state $x$ ($n = 18$) and the input $u$ ($p = 6$) are

$$x = \begin{pmatrix} \rho^T & \dot{\rho}^T & \mathcal{F}^T \end{pmatrix}^T \qquad u = \Delta\mathcal{F}. \qquad (32)$$

$A^{cm}$, $B^{cm}$ are obtained by discretizing (31) with a zero order hold and a sampling time of 0.5.

The limitations of the actuators are

$$-0.5 \le \Delta\mathcal{F}_i(t_k) \le 0.5 \qquad -1 \le \mathcal{F}_i(t_k) \le 1. \qquad (33)$$

So we have $q = r = 6$. We choose the weighting matrices as $Q = I$, $R = I$ and $T = I$ and use a horizon of $N = 30$.

We investigate the algorithm for $c = 100$ and using $i^{ALM}_{Max} = 4$ and $i^{FGM}_{Max} = 40$ iterations for the subproblem and warm-starting. This results in a computation time of about 7.4 ms using a single core of a 2.4 GHz Intel Q6600 CPU.

To obtain an accuracy estimate we use Monte Carlo simulations and compare the obtained solution with the exact solution. We assume that the chain is at $t_0$ at rest at initial positions randomly chosen from a uniform distribution: $\rho_i(t_0) \in [-5, 5]$. Moreover, we apply on each mass in each sampling step a random force uniformly distributed between $-2$ and 2. We compare the difference in the accumulated cost and the input difference

$$\chi = \frac{\sum_k |J^s(x(t_k), u(t_k)) - J^s(\tilde{x}(t_k), \tilde{u}(t_k))|}{\sum_k J^s(x(t_k), u(t_k))} \qquad (34a)$$

$$\psi = \frac{\sum_k \|u(t_k) - \tilde{u}(t_k)\|}{\sum_k \|u(t_k)\|}, \qquad (34b)$$

where $u$, $x$ correspond to exact solution of (9) using Matlab's active set solver *quadprog* and $\tilde{u}$, $\tilde{x}$ denote the solution using the proposed algorithm. Note that we use the same realization of the random forces and initial positions for comparison.

Figure 4 illustrates a sample path behavior. Note that the difference between exact and inexact solution is barely visible and that both type of constraints (33) are at some time instances active.

Using 20 such simulations each 6000 steps long and evaluating (34) over the last 5000 steps we obtain the averaged values $\chi_{avg} = 0.06\%$ and $\psi_{avg} = 0.15\%$.

Table II compares the performance of the proposed algorithm with our algorithm presented in [9], qpOASES [5] and the algorithm [22].

For *qpOASES* we limit the number of working set changes to 30 to guarantee a maximum computation time. For the algorithm [22] we set the maximum number of Newton iterations to 10 and choose after some tuning as barrier parameter $10^{-2}$. Since this algorithm solves problems with box-constraints we limit $\rho_i$ and $\dot{\rho}_i$, $i = 1, \ldots, 6$ to $\pm 10^{15}$. Note that these dummy constraints are never active during the simulations. We observe that our proposed algorithm obtains faster and more accurate results.

[5] A video demonstration of this system is online available at the webpage http://ifatwww.et.uni-magdeburg.de/syst/about_us/people/zometa/.

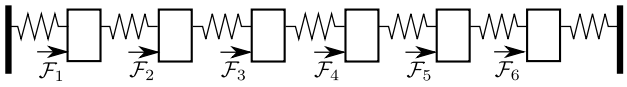


Fig. 3. Benchmark example: Chain of masses connected by springs.

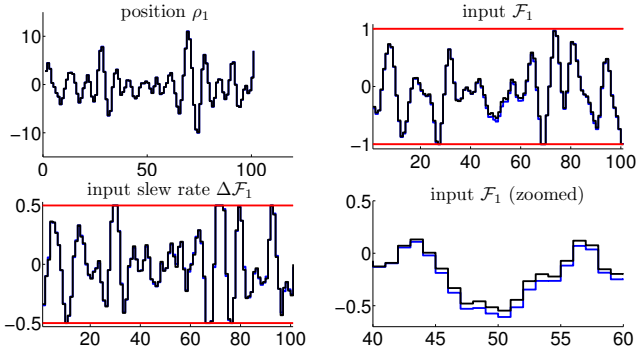


Fig. 4. Sample paths. Black: exact solution. Blue: inexact solution. Red: actuator limitations (constraints).

| Algorithm | $\chi_{avg}$ | $\psi_{avg}$ | $T_{max}$ | $T_{avg}$ |
|---|---|---|---|---|
| Proposed algorithm | 0.06% | 0.15% | 7.4ms | 7.4ms |
| Algorithm [9] | 0.07% | 0.18% | 15ms | 15ms |
| qpOASES [5] | 0.68% | 0.49% | 24ms | 7.9ms |
| Algorithm [22] | 0.07% | 0.31% | 18ms | 11ms |

TABLE II

PERFORMANCE FOR BENCHMARK EXAMPLE. $T_{max}$ / $T_{avg}$ MAXIMUM / AVERAGE COMPUTATION TIME, $\chi_{avg}$ COST DIFFERENCE / SUBOPTIMALITY INDEX, $\psi_{avg}$ INPUT DIFFERENCE.

Finally, let us compare the proposed algorithm with [9]. To obtain similar accuracy we used the same penalty parameter $c$ and number of multiplier iterations and 80 iterations to solve the subproblem. The required computation time of [9] is about twice as large, which is mainly due to the increased number of iterations. For this case the proposed algorithm requires only about 3% less time per iteration of Nesterov's method than [9], as discussed in Section V-E.

Overall, the proposed algorithm delivers for this example good performance.

## VII. SUMMARY AND FUTURE WORK

In this work we presented an iterative algorithm for predictive control of linear systems with constraints, which combines Nesterov's gradient method and the method of multipliers. In particular, we outlined how the nonsmooth subproblem can be solved with Nesterov's gradient method and how to compute the necessary constants. We discussed implementation issues of the algorithm and illustrated its performance by a benchmark example. Additionally, a real implementation demonstrated its suitability for low cost embedded systems.

Future work will focus on further evaluation of the proposed algorithms, an analytical investigation of the overall convergence and further comparisons with other algorithms.

## APPENDIX

### A. Matrix $E$ and Vectors $\overline{\mathbf{z}}(x(t_k))$, $\underline{\mathbf{z}}(x(t_k))$

The matrix $E \in \mathbb{R}^{Nq+r\times Np}$ and vectors $\overline{\mathbf{z}}(x(t_k))$, $\underline{\mathbf{z}}(x(t_k))$ can represent state constraints, input constraints or mixed constraints. $E$ and $\overline{\mathbf{z}}(x(t_k))$ are given by

$$\overline{\mathbf{z}}(x(t_k)) = \begin{pmatrix} \overline{e}-CA^0x(t_k) \\ \vdots \\ \overline{e}-CA^{N-1}x(t_k) \\ \overline{f}-FA^Nx(t_k) \end{pmatrix} \tag{35}$$

$$E = \begin{pmatrix} D & 0 & \dots & 0 \\ CB & D & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N-2}B & CA^{N-3}B & \dots & D \\ FA^{N-1}B & FA^{N-2}B & \dots & FB \end{pmatrix} \tag{36}$$

and $\underline{\mathbf{z}}(x(t_k))$ is similar to $\overline{\mathbf{z}}(x(t_k))$.

### B. System matrices for 2-link robotic arm

The continuous-time matrices are

$$A^c = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & -17.2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -16.1 \end{pmatrix}, B^c = \begin{pmatrix} 0 & 0 \\ 2.62 & 0 \\ 0 & 0 \\ 0 & 2.48 \end{pmatrix}.$$

The state and input vectors are

$$x^T = \begin{pmatrix} \theta_1 & \omega_1 & \theta_2 & \omega_2 \end{pmatrix}, u^T = \begin{pmatrix} u_1 & u_2 \end{pmatrix}.$$

The weighting matrices $Q$, $R$, $S$ and $T$ in (7) are chosen as

$$Q = \operatorname{diag}\left( 1.14\text{x}10^4 \;\; 2.24\text{x}10^1 \;\; 1.40\text{x}10^4 \;\; 2.94\text{x}10^1 \right),$$

$$R = \operatorname{diag}\left( 2.20\text{x}10^{-1} \;\; 2.37\text{x}10^{-1} \right), S = 0, T = Q.$$

### C. Proof of Proposition 1

We need to show that $L$ and $\phi$ satisfy (21) and (22), respectively for all $\mathbf{u} \in \mathbb{R}^{Np}$ and $\mathbf{v} \in \mathbb{R}^{Np}$.

Let $\mathbf{u}$ and $\mathbf{v}$ with $\mathbf{u} \neq \mathbf{v}$ be arbitrary and let the segment connecting $\mathbf{u}$ with $\mathbf{v}$ be given by

$$\Gamma(\zeta) = \mathbf{u} + \zeta(\mathbf{u} - \mathbf{v}), \zeta \in [0, 1] \tag{37}$$

and let $\sigma = \mathbf{u} - \mathbf{v}$.

Let us show that although the function $f^a$ (15) is not everywhere twice differentiable, $f^a$ has on $\Gamma$ only at a finite number of points no second directional derivative in the direction $\frac{\sigma}{\|\sigma\|}$. From (17), (19) we observe that each part $P_j$ of the function $P$ consists of three pieces and a switching between the pieces appears at $\Gamma(\zeta)$, if the sign of $\mathcal{S}_j^+$ or of $\mathcal{S}_j^-$ changes, i.e., at $\zeta$ we need to have

$$\mathcal{S}_j^+(\Gamma(\zeta)) = 0 \quad \mathcal{S}_j^+(\Gamma(\zeta+\epsilon))\mathcal{S}_j^+(\Gamma(\zeta-\epsilon)) < 0, \tag{38}$$

or

$$\mathcal{S}_j^-(\Gamma(\zeta)) = 0 \quad \mathcal{S}_j^-(\Gamma(\zeta+\epsilon))\mathcal{S}_j^-(\Gamma(\zeta-\epsilon)) < 0, \tag{39}$$

for all $\epsilon > 0$ [6]. Note that $\mathcal{S}_j^+(\Gamma(\zeta))$ and $\mathcal{S}_j^-(\Gamma(\zeta))$ are linear in $\zeta$ since $\mathcal{S}_j^+(\mathbf{u})$ and $\mathcal{S}_j^-(\mathbf{u})$ are linear in $\mathbf{u}$ and $\Gamma$ is linear in $\zeta$. Thus for each $j$ there can be maximally one $\zeta$ such that (38) holds and one $\zeta$ such that (39) holds.

[6]We display in this proof only the dependence of $\mathcal{S}^+$, $\mathcal{S}_j^-$, $P$, $P_j$ and $f^a$ on $\mathbf{u}$ and skip the dependence on the (constant) $\xi$ and $x(t_k)$.

Overall, there can be maximally $\Theta \leq 2(Nq+r)$ points on $\Gamma$ where (38) or (39) hold for some $j$. Let us denote these points by $\Gamma(\omega_1), \Gamma(\omega_2), \ldots, \Gamma(\omega_\Theta)$ with $\omega_1 < \omega_2 \ldots < \omega_\Theta$. Note that we have $0 \leq \omega_1$ and $\omega_\Theta \leq 1$, compare (37).

If we define additionally $\omega_0 = 0$, $\omega_{\Theta+1} = 1$, then we have due to (37) $\sum_{i=0}^{\Theta}(\omega_{i+1} - \omega_i) = 1$ and

$$\nabla f^a(\mathbf{u}) - \nabla f^a(\mathbf{v}) = \sum_{i=0}^{\Theta} \nabla f^a(\Gamma(\omega_{i+1})) - \nabla f^a(\Gamma(\omega_i)). \tag{40}$$

From (25) follows that the directional derivative of $f^a$ in the direction $\frac{\sigma}{\|\sigma\|}$ is linear, if $\zeta \in (\omega_i, \omega_{i+1})$ for some $0 \leq i \leq \Theta$.

Thus we have

$$\nabla f^a(\Gamma(\omega_{i+1})) - \nabla f^a(\Gamma(\omega_i)) = \mathcal{M}_i \sigma(w_{i+1} - w_i) \tag{41a}$$

$$\mathcal{M}_i = H + \sum_{j=1}^{Nq+r} \delta_j c E_j^T E_j. \tag{41b}$$

where $\delta_j = 0$ if $\mathcal{S}_j^-(\Gamma(\zeta)) \geq 0$ and $\mathcal{S}_j^+(\Gamma(\zeta)) \leq 0$ for $\zeta \in (\omega_i, \omega_{i+1})$ and $\delta_j = 1$ otherwise.

Since the matrices $H$ and $cE_j^T E_j$, $j = 1, \ldots, Nq+r$ are symmetric and positive semi-definite, we can bound $\|\mathcal{M}_i\|$ independent of $i$ by

$$\|\mathcal{M}_i\| \leq \|H + c \sum_{j=1}^{Nq+r} E_j^T E_j\| = \|H + cE^T E\| = L. \tag{42}$$

In combination with (40), (41) we obtain

$$\|\nabla f^a(\mathbf{u}) - \nabla f^a(\mathbf{v})\| \leq L\|\mathbf{u} - \mathbf{v}\|. \tag{43}$$

Hence, (21) holds for $\mathbf{u}$ and $\mathbf{v}$ with $L = \|H + cE^T E\|$. To verify (22) note that from (40), (41)

$$\langle \nabla f^a(\mathbf{u}) - \nabla f^a(\mathbf{v}), \sigma \rangle = \sum_{i=0}^{\Theta} (\omega_{i+1} - \omega_i) \langle \mathcal{M}_i \sigma, \sigma \rangle. \tag{44}$$

Since for each $i$ $\mathcal{M}_i$ is a sum of symmetric, positive semi-definite matrices we have

$$\langle \mathcal{M}_i \sigma, \sigma \rangle \geq \lambda_{Min}(\mathcal{M}_i)\|\sigma\|^2 \geq \lambda_{Min}(H)\|\sigma\|^2. \tag{45}$$

Combining this with (44) yields

$$\langle \nabla f^a(\mathbf{u}) - \nabla f^a(\mathbf{v}), \mathbf{u} - \mathbf{v} \rangle \geq \lambda_{Min}(H)\|\mathbf{u} - \mathbf{v}\|^2. \tag{46}$$

So (22) holds for $\mathbf{u}$ and $\mathbf{v}$ with $\phi = \lambda_{Min}(H)$.

Since $\mathbf{u}$, $\mathbf{v}$ were arbitrary, (22), (21) hold for every $\mathbf{u}$, $\mathbf{v}$.

### *D. Proof of Proposition 2*

Note that we have

$$L = \|H + cE^T E\|, \tag{47}$$

and $H$, $E^T E$ are symmetric, positive semi-definite. Thus

$$\|H\| + c\|E^T E\| \geq L \geq \max(\|H\|, c\|E^T E\|), \tag{48}$$

which verifies a). Since $\phi = \lambda_{Min}(H)$ is independent of $c$ part b) follows from (47) and the continuity properties of eigenvalues. Finally, $\phi = \lambda_{Min}(H)$ and a) yield c).

## REFERENCES


[1] BEMPORAD, A., MORARI, M., DUA, V., AND PISTIKOPOULOS, E. The explicit linear quadratic regulator for constrained systems. *Automatica 38*, 1 (2002), 3–20.

[2] BERTSEKAS, D. P. *Constrained Optimization and Lagrange multiplier methods*. Athena Scientific, 1996.

[3] DEVOLDER, O., GLINEUR, F., AND NESTEROV, Y. First-order Methods of Smooth Convex Optimization with Inexact Oracle. Available online at http://www.optimization-online.org.

[4] FERNANDEZ, D. SOLODOV, M. Local convergence of exact and inexact augmented Lagrangian methods under the second-order sufficient optimality condition. SIAM Journal on Optimization, To Appear 2012.

[5] 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* (2008), 816–830.

[6] GARCIA, C., PRETT, D., AND MORARI, M. Model predictive control: Theory and practice - A survey. *Automatica 25*, 3 (1989), 335–348.

[7] KÖGEL, M., AND FINDEISEN, R. A Fast Gradient method for embedded linear predictive control. In *Proc. of the IFAC World Congress 2011* (2011), pp. 1362–1367.

[8] KÖGEL, M., AND FINDEISEN, R. Fast predictive control of linear systems combining Nesterov's gradient method and the method of multipliers. In *Proc. IEEE Conference on Decision and Control and ECC* (2011), pp. 501–506.

[9] 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 Multi-conference on Systems and Control* (2011), pp. 780–785.

[10] MACIEJOWSKI, J. M. *Predictive Control with Constraints*. Prentice Hall, Upper Saddle River, New Jersey, 2002.

[11] MANCUSO, G., AND KERRIGAN, E. Solving Constrained LQR Problems by Eliminating the Inputs from the QP. In *Proc. IEEE Conference on Decision and Control and ECC* (2011), pp. 507–512.

[12] MATTINGLEY, J., WANG, Y., AND BOYD, S. Receding horizon control: Automatic generation of high-speed solvers. *IEEE Control Systems Magazine 31*, 3 (2011), 52–65.

[13] MAYNE, D. Q., RAWLINGS, J. B., RAO, C. V., AND SCOKAERT, P. Constrained model predictive control: Stability and optimality. *Automatica 36* (2000), 789–814.

[14] MILMAN, R., AND DAVISON, E. J. A fast MPC algorithm using nonfeasible active set methods. *Journal of Optimization Theory and Applications 139* (2008), 591–616.

[15] NESTEROV, Y. A method for solving a convex programming problem with convergence rate $1/k^2$. *Soviet Mathematics Doklady 27*, 2 (1983), 372–376.

[16] NESTEROV, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Acad. Publ., 2004.

[17] QIN, S., AND BADGWELL, T. A survey of industrial model predictive control technology. *Control engineering practice 11*, 7 (2003), 733–764.

[18] RAO, C., WRIGHT, S., AND RAWLINGS, J. Application of Interior-Methods to Model Predictive Control. *Journal of Optimization Theory and Applications 99* (1998), 723–757.

[19] RICHTER, S., JONES, C. N., AND MORARI, M. Computational Complexity Certification for Real-Time MPC with Input Constraints Based on the Fast Gradient Method. IEEE Transactions on Automatic Control, To Appear 2012.

[20] 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. IEEE Conference on Decision and Control and ECC* (2011), pp. 5223–5229.

[21] ROCKAFELLAR, R. T. The Multiplier Method of Hestenes and Powell Applied to Convex Programming. *Journal of Optimization Theory and Applications 12*, 6 (1973), 555–562.

[22] WANG, Y., AND BOYD, S. Fast Model Predictive Control Using Online Optimization. *IEEE Transactions on Control Systems Technology 18*, 2 (2010), 267–278.

[23] WASCHL, H., ALBERER, D., AND DEL RE, L. Numerically Efficient Self Tuning Strategies for MPC of Integral Gas Engines. In *Proceedings of the 18th IFAC World Congress* (2011), pp. 2482–2487.

[24] ZOMETA, P., KÖGEL, M., FAULWASSER, T., AND FINDEISEN., R. Implementation Aspects of Model Predictive Control for Embedded Systems. ACC2012, To Appear. Preprint available under http://ifatwww.et.uni-magdeburg.de/syst/about_us/people/zometa/.