Solving LP-MPC problems using ramp functions*

Morten Hovd, Giorgio Valmorbida

Solving LP-MPC problems using ramp functions*

Morten Hovd[1] and Giorgio Valmorbida[2]

***Abstract*— A new implementation of the Simplex method for solving linear programming problems is developed, and its application for solving MPC problems with linear objective functions is described. A detailed comparison with existing implementations of the Simplex method is beyond the scope of this paper. However, initial numerical results indicate that gains in calculation times may be achieved for problems of small to moderate size.**

I. INTRODUCTION

Model Predictive Control (MPC) has been a great success in industry [1], and since its initial development in the 1970's (see, *e.g.* [2]) it has found application in a wide range of industrial processes. The main feature of MPC distinguishing it from classical control design methods such as Linear Quadratic Regulator is the ability to take into account constraints in both inputs, state, and outputs. However, in the conventional MPC formulation, an optimization problem has to be solved online, thus incurring a significant computational cost. Many approaches have been studied in order to reduce the computational requirements of MPC, including input blocking [3] and utilizing structure in the optimization problem (*e.g.* [4]). This paper will make no attempt at covering all such approaches. Instead, the ramp-based approach to solving quadratic programming (QP) problems, proposed recently in [5], is extended to linear programming (LP) problems.

The next section describes the LP-MPC formulation used. Thereafter, the main result of the paper, the ramp-based solution to LP problems in standard form, is presented, followed by a description of how the Simplex method can be used with the ramp-based solution. Numerical experiments are performed to compare the calculation performance of the new approach to that of of the well known commercial solver CPLEX[6] from IBM. Initial results indicate computational advantages for small to moderate-size problems.

II. LP-MPC FORMULATION

A linear programming problem in standard for can be expressed as

$$\begin{aligned} &\text{minimize}_z \;\; c^\top z && \text{(1a)}\\ &\text{subject to} \\ &\qquad Gz = b && \text{(1b)}\\ &\qquad -z \leq 0 && \text{(1c)} \end{aligned}$$

where $G \in \mathbb{R}^{n_e \times n_z}$, $c \in \mathbb{R}^{n_z \times 1}$, $z \in \mathbb{R}^{n_z \times 1}$, and $b \in \mathbb{R}^{n_e \times 1}$. Naturally it is assumed that $n_e < n_z$, as otherwise there would be no degrees of freedom for optimization.

The system dynamics are given by the linear discrete-time state space model

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) && \text{(2a)}\\ y(k) &= Cx(k) && \text{(2b)} \end{aligned}$$

The constraints on the inputs u and outputs y are given by polytopic sets[1] of the form

$$\begin{aligned} y_{min} \leq y \leq y_{max} && \text{(3a)}\\ u_{min} \leq u \leq u_{max}. && \text{(3b)} \end{aligned}$$

We wish to compute the input sequence that minimizes the cost

$$J(\mathbf{y}, \mathbf{u}, x(k+N)) = \|Px(k+N)\|_p + \sum_{i=1}^{N-1} \|Qy(k+i)\|_p + \|Ru_{k+i-1}\|_p \quad \text{(4)}$$

where $\mathbf{y}$ and $\mathbf{u}$ denote the concatenated vectors $\mathbf{y} = [y(k+1)^\top \;\; y(k+2)^\top \;\; \cdots \;\; y(k+N)^\top]^\top$ and $\mathbf{u} = [u(k)^\top \;\; u(k+1)^\top \;\; \cdots \;\; u(k+N-1)^\top]^\top$, respectively, and $Q \in \mathbb{R}^{n_c \times n_y}$, $R \in \mathbb{R}^{n_u \times n_u}$. To have a sensible LP-MPC formulation, it is naturally assumed that (A, B) is stabilizable and $(Q^{\frac{1}{2}}C, A)$ is detectable and R full rank.

The purpose of this paper is not the LP-MPC solution *per se*, but rather to present a numerical solution strategy to compute the input values $\mathbf{u}$. Although terminal sets and costs are important for recursive feasibility and stability, and systematic approaches to calculating these do exist also for LP-MPC [7], they will for simplicity be ignored in the following, as they are not important for presenting the contributions of the paper. Furthermore, $p = 1$ will be assumed throughout the paper, although LP-MPC using $p = \infty$ can also be formulated in a straight forward way.

*This work was supported by the French-Norwegian exchange programme AURORA under contract 294676.

[1]Morten Hovd is with the Department of Engineering Cybernetics, Norwegian University of Science & Technology, 7491 Trondheim, Norway. morten.hovd@itk.ntnu.no

[2]Giorgio Valmorbida is with L2S, Laboratoire des Signaux et Systèmes, CentraleSupelec, CNRS, Université Paris-Saclay, 3 rue Joliot Curie, Gif-sur-Yvette, 91192 France. giorgio.valmorbida@centralesupelec.fr

[1]One may have constraints on both x and y, but often it is more natural to specify the constraints in terms of y. If there are state constraints that do not directly correspond to a constraint on an output, the output vector can simply be augmented with one or more 'dummy outputs'.

In the following, it will be described how to translate the objective function (4), dynamics (2a,2b) and constraints (3a,3b) into an LP problem formulation on the form (1a)-(1c). This development is rather standard, but the presented level of detail is necessary to present the LP-MPC solution procedure that is the main contribution of this paper. Repeated use of the system dynamics dynamics (2a,2b) allow us to express the future measurements as

$$\mathbf{y} = C_A x(k) + C_B \mathbf{u} \tag{5}$$

where

$$A_0 = \begin{bmatrix} A \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$
$$\tilde{B} = \text{diag}\{B\}$$
$$\tilde{C} = \text{diag}\{C\}$$
$$I_A = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -A & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -A & I & 0 \\ 0 & \cdots & 0 & -A & I \end{bmatrix}$$

and $C_A = \tilde{C} I_A^{-1} A_0$ and $C_B = \tilde{C} I_A^{-1} \tilde{B}$. The output constraints trivially translate to

$$\mathbf{y} \leq \underline{1}_N \otimes y_{max} \tag{6a}$$
$$-\mathbf{y} \leq -\underline{1}_N \otimes y_{min}, \tag{6b}$$

where $\underline{1}_N$ denotes a length N column vector of ones, and $\otimes$ denotes the Kronecker product. Similarly, the input constraints are expressed as

$$\mathbf{u} \leq \underline{1}_N \otimes u_{max} \tag{7a}$$
$$-\mathbf{u} \leq -\underline{1}_N \otimes u_{min}. \tag{7b}$$

In order to translate the objective function in (4) with $p = 1$ into the form used in (4), we split y and u into positive and negative parts

$$\mathbf{y} = \mathbf{y_p} - \mathbf{y_m}$$
$$\mathbf{u} = \mathbf{u_p} - \mathbf{u_m}.$$

Introducing auxiliary variables $\mathbf{s_p}, \mathbf{s_m}, \mathbf{s_{up}}, \mathbf{s_{um}}$, the equality constraint (5) together with inequality constraints (6a-7b) are converted to the equality constraint (1b), with

$$G = \begin{bmatrix} I & -I & -C_B & C_B & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & -I & 0 & 0 & 0 & -I & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & -I & 0 & 0 & 0 & I \end{bmatrix} \tag{8}$$

$$b = \begin{bmatrix} C_A x(k) \\ \underline{1}_N \otimes y_{max} \\ \underline{1}_N \otimes y_{min} \\ \underline{1}_N \otimes u_{max} \\ \underline{1}_N \otimes u_{min} \end{bmatrix} \tag{9}$$

where

$$z = [\mathbf{y_p}^\top \ \mathbf{y_m}^\top \ \mathbf{u_p}^\top \ \mathbf{u_m}^\top \ \mathbf{s_p}^\top \ \mathbf{s_m}^\top \ \mathbf{s_{up}}^\top \ \mathbf{s_{um}}^\top]^\top$$

in (1b) and (1c).

The objective function now follows from (1a) with

$$c^\top = \begin{bmatrix} c_y^\top & c_y^\top & c_u^\top & c_u^\top & 0 & 0 & 0 & 0 \end{bmatrix}$$

where

$$c_y = \underline{1}_N^\top \otimes \sigma(Q)$$
$$c_u = \underline{1}_N^\top \otimes \sigma(R)$$

and $\sigma(Q)$ and $\sigma(R)$ are row vectors of column sums of Q and R, respectively, such that, for $p = 1$, we obtain

$$J(\mathbf{y}, \mathbf{u}, x(k+N)) = c^\top z.$$

III. Main results

This section first provides some background on ramp functions, as presented in [5]. Next, it is shown how a Linear Complementarity Problem (LCP) can be associated to an LP to obtain a PWA representation implicitly defined in terms of ramp functions. Section IV details how this implicit PWA representation can be exploited to quickly calculate solutions to the LP-MPC problem.

A. Ramp functions

Definition 1: The ramp function $r(y)$ is given by

$$r(y) = \begin{cases} 0 & \text{if } y < 0 \\ y & \text{if } y \geq 0 \end{cases} \tag{10}$$

Lemma 1: The ramp function is the only function satisfying, $\forall y \in \mathbb{R}$

$$(r(y) - y) r(y) = 0 \tag{11a}$$
$$r(y) \geq 0 \tag{11b}$$
$$(r(y) - y) \geq 0. \tag{11c}$$

Proof: First note that the ramp function can be expressed as the unique solution to the convex optimization problem parameterized in y (thus depending on y) with strictly convex objective function as follows

$$\underset{r}{\text{minimize}} \ \frac{1}{2}(r-y)^2 \quad \text{subject to} \quad r \geq 0. \tag{12}$$

With the Lagrangian associated to the optimization problem, $\mathcal{L}(r, \lambda) = \frac{1}{2}(r-y)^2 - \lambda r$, we obtain the Karush-Kuhn-Tucker (KKT) conditions

$$(r - y) - \lambda = 0; \ \lambda r = 0; \ r \geq 0; \ \lambda \geq 0$$

which are necessary for optimality and also sufficient since the problem is strictly convex. To obtain a description in the variables (y, r) one can use $\lambda = (r - y)$ above to obtain $r \geq 0, \quad (r - y) \geq 0, \quad r(r - y) = 0.$ ■

For $y \in \mathbb{R}^m$ let us define the function $\mathbf{r} : \mathbb{R}^m \to \mathbb{R}^m$, the vector-valued ramp function, as

$$\mathbf{r}(y) = \begin{bmatrix} r(y_1) & r(y_2) & \cdots & r(y_m) \end{bmatrix}^\top$$

Remark 1: Note that, due to the piecewise definition of the ramp function in (10), we have that vector $\mathbf{r}$ can also be expressed as the product $\mathbf{r}(y) = I_{\mathcal{A}} y$ where $I_{\mathcal{A}} \in \mathbb{D}^m$, with its diagonal elements verifying $I_{\mathcal{A}(i,i)} \in \{0, 1\}$. Clearly, the set $\mathcal{A} \subseteq \{1, 2, \ldots, N\}$ depends on y. ⌟

B. Expressing the solution to an LP problem using ramp functions

Theorem 1: The solution of the LP-MPC is a PWA function given by

$$\left\{ \begin{array}{rcl} \mathbf{u}(x) & = & M_1 x + M_2 \mathbf{r}(y) + m_0 \\ y & = & M_3 x + M_4 \mathbf{r}(y) + m_5 \end{array} \right. \tag{14}$$

Proof: From the Lagrangian function $\mathcal{L}(z, \lambda) = c^\top z + \mu^\top (G\mathbf{z} - b) - \lambda^\top z$, we get the KKT conditions

$$c + G^\top \mu - \lambda = 0 \tag{15a}$$
$$Gz = b \tag{15b}$$
$$-z \leq 0 \tag{15c}$$
$$\lambda \geq 0 \tag{15d}$$
$$\lambda_i z_i = 0 \ \forall i \tag{15e}$$

With the matrices and vectors partitioned (and reordered as necessary) as

$$G = \begin{bmatrix} G_a & G_b \end{bmatrix}, \ \lambda = \begin{bmatrix} \lambda_a \\ \lambda_b \end{bmatrix}, \ z = \begin{bmatrix} z_a \\ z_b \end{bmatrix}$$

with square and invertible G_b. Set

$$\begin{bmatrix} \lambda_a \\ \lambda_b \end{bmatrix} = \begin{bmatrix} r(y_a) - y_a \\ r(y_b) \end{bmatrix}, \tag{16}$$

we can reformulate the KKT conditions as

$$c_p + \begin{bmatrix} G_a^\top & -I_a \\ G_b^\top & 0 \end{bmatrix} \begin{bmatrix} \mu \\ \lambda_a \end{bmatrix} - \begin{bmatrix} 0 \\ I_b \end{bmatrix} \lambda_b = 0 \tag{17a}$$
$$\begin{bmatrix} G_a & G_b \end{bmatrix} \begin{bmatrix} z_a \\ z_b \end{bmatrix} = b \tag{17b}$$
$$\begin{bmatrix} z_a \\ z_b \end{bmatrix} \geq 0 \tag{17c}$$
$$\begin{bmatrix} \lambda_a \\ \lambda_b \end{bmatrix} \geq 0 \tag{17d}$$
$$\begin{bmatrix} \lambda_a \\ \lambda_b \end{bmatrix} \times \begin{bmatrix} z_a \\ z_b \end{bmatrix} = 0 \tag{17e}$$

where the subscript p on c_p indicates that c has to be reordered after partitioning z. In contrast, it is only columns of G that are reordered, and hence there is no need for reordering vector b.

With the ramp functions assigned to λ_a, λ_b as in (16), we associate the complementarity conditions (15e) with the complementarity (11a) of the ramp function. Therefore, we obtain for z_a and for z_b

$$\lambda_{ai} z_{ai} = (r(y_a) - y_a)_i r(y_a)_i = 0 \Rightarrow r(y_a) = z_a \tag{18a}$$
$$\lambda_{bi} z_{bi} = r(y_b)_i (r(y_b) - y_b)_i = 0 \Rightarrow r(y_b) - y_b = z_b. \tag{18b}$$

Define

$$\Xi = \begin{bmatrix} G_a^\top & -I_a \\ G_b^\top & 0 \end{bmatrix}^{-1}$$

which is partitioned as

$$\begin{bmatrix} \Xi_1 \\ \Xi_2 \end{bmatrix} = \begin{bmatrix} 0 & G_b^{-T} \\ -I_a & G_a^\top G_b^{-T} \end{bmatrix}.$$

We obtain from (17a)

$$\begin{bmatrix} \mu \\ r(y_a) - y_a \end{bmatrix} = \Xi \left(-c_p + \begin{bmatrix} 0 \\ I_b \end{bmatrix} r(y_b) \right)$$

from which we obtain

$$\mu = -\Xi_1 c_p + \Xi_1 \begin{bmatrix} 0 \\ I_b \end{bmatrix} r(y_b) \tag{19a}$$
$$y_a = \Xi_2 c_p - \Xi_2 \begin{bmatrix} 0 \\ I_b \end{bmatrix} r(y_b) + r(y_a) \tag{19b}$$

From (17b) we have

$$G_a r(y_a) + G_b (r(y_b) - y_b) = b$$

and since G_b is invertible this can be expressed as

$$y_b = r(y_b) + G_b^{-1} G_a r(y_a) - G_b^{-1} b \tag{20}$$

In summary, we obtain

$$z = \begin{bmatrix} z_a \\ z_b \end{bmatrix} = \begin{bmatrix} r(y_a) \\ r(y_b) - y_b \end{bmatrix} \tag{21a}$$
$$\lambda = \begin{bmatrix} \lambda_a \\ \lambda_b \end{bmatrix} = \begin{bmatrix} r(y_a) - y_a \\ r(y_b) \end{bmatrix} \tag{21b}$$
$$\mu = -\Xi_1 c_p + \Xi_1 \begin{bmatrix} 0 \\ I_b \end{bmatrix} r(y_b) \tag{21c}$$
$$\begin{aligned} \begin{bmatrix} y_a \\ y_b \end{bmatrix} &= \begin{bmatrix} -I_a & G_a^\top G_b^{-\top} \\ 0 & 0 \end{bmatrix} c_p - \begin{bmatrix} 0 \\ G_b^{-1} \end{bmatrix} b \\ &\quad + \begin{bmatrix} I & -G_a^\top G_b^{-\top} \\ G_b^{-1} G_a & I \end{bmatrix} \begin{bmatrix} r(y_a) \\ r(y_b) \end{bmatrix} \\ &= M_c c_p + M_b b + M_r \begin{bmatrix} r(y_a) \\ r(y_b) \end{bmatrix} \end{aligned} \tag{21d}$$

Noting that for the MPC formulation above we have

$$b = \begin{bmatrix} C_A x(k) \\ y_{max} \\ y_{min} \\ u_{max} \\ u_{min} \end{bmatrix} = \begin{bmatrix} C_A \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ y_{max} \\ y_{min} \\ u_{max} \\ u_{min} \end{bmatrix} = b_x x + b_0$$

therefore, (21d) becomes

$$\begin{bmatrix} y_a \\ y_b \end{bmatrix} = M_3 x + M_4 \begin{bmatrix} r(y_a) \\ r(y_b) \end{bmatrix} + m_5 \tag{22a}$$

$$M_3 = -\begin{bmatrix} 0 \\ G_b^{-1}(:, 1 : n \cdot n_y) \end{bmatrix} C_A \tag{22b}$$

$$M_4 = \begin{bmatrix} I & -G_a^\top G_b^{-\top} \\ G_b^{-1} G_a & I \end{bmatrix} \tag{22c}$$

$$m_5 = \begin{bmatrix} -I_a & G_a^\top G_b^{-\top} \\ 0 & 0 \end{bmatrix} c_p - \begin{bmatrix} 0 \\ G_b^{-1} \end{bmatrix} \begin{bmatrix} 0 \\ y_{max} \\ y_{min} \\ u_{max} \\ u_{min} \end{bmatrix} \tag{22d}$$

where $G_b^{-1}(:, 1 : n \cdot ny)$ indicates that we only retain the $n \cdot n_y$ leading columns of G_b^{-1}. From (18a) and (18b) we obtain

$$z = \begin{bmatrix} I & 0 \\ -G_b^{-1} G_a & 0 \end{bmatrix} \begin{bmatrix} r(y_a) \\ r(y_b) \end{bmatrix} + G_b^{-1}(b_x x + b_0)$$

and noting that

$$\mathbf{u} = \begin{bmatrix} 0 & 0 & I & -I & 0 & 0 & 0 & 0 \end{bmatrix} z = P_z z$$

we get

$$M_1 = -P_z M_3 = P_z \begin{bmatrix} 0 \\ G_b^{-1}(:, 1 : n \cdot n_y) \end{bmatrix} C_A \tag{23a}$$

$$M_2 = P_z \begin{bmatrix} I & 0 \\ -G_b^{-1} G_a & 0 \end{bmatrix} \tag{23b}$$

$$m_0 = P_z G_b^{-1} b_0 \tag{23c}$$

■

IV. Implementation

In this section, we will first present a brief recap of the Simplex method for solving LPs, and next detail how the Simplex method can be implemented via the ramp-based description developed above.

A. The Simplex method

In the Simplex method, the decision variables z are partitioned into n_e *basis* variables z_B and $n_z - n_e$ variables z_0 that are set to zero. To the variables z_0 (corresponding to active inequality constraints) we can associate Lagrange multipliers λ_0. We have from (1b) that $z_B = G_B^{-1} b$, where G_B consists of the columns of G corresponding to z_B. A solution for which $z_B \geq 0$ is termed a *basic feasible solution*. If a Lagrange variable $\lambda_{0i} < 0$, this indicates that the objective function can be reduced by allowing z_{0i} to increase from 0, *i.e.*, by allowing z_{0i} to enter the set of basis variables. If there are more than one negative Lagrange multiplier, we will assume that the first to enter the set of basis variables is the one with the most negative value. The corresponding variable to leave the active set is the first variable to become zero as variable entering the active set is increased. We find that the first basis variable to become zero is the variable z_{Bj} corresponding to the minimum value of

$$\delta_j = \frac{z_{B_j}}{G^{-1}_{B(j,:)} G_{(:,i)}} = \frac{z_{B_j}}{\alpha_j}$$

The division indicated above need only be performed if $\alpha_j > 0$. The Simplex method assumes that a basic feasible solution is available. In each iteration, new basis variables and new Lagrange multipliers are calculated. An optimal solution is found if all Lagrange multipliers are non-negative.

B. A ramp-based implementation of the Simplex method

To recap, we are looking for solutions to (35):

$$\begin{bmatrix} y_a \\ y_b \end{bmatrix} = M_c c_p + M_b b + M_r \begin{bmatrix} r(y_a) \\ r(y_b) \end{bmatrix}$$

We will (with some abuse of concept) call the set of variables y for which $y = r(y)$ as the *active set*, denoted $\mathcal{A}$, and the set of inactive variables, for which $r(y) = 0$, denoted $\mathcal{A}^C$. A solution to the LP is found if $y_i > 0 \; \forall i \in \mathcal{A}$ and $y_i \leq 0 \forall i \in \mathcal{A}^C$. We also have the set of variables $\mathbf{A}$ for the y's corresponding to z_a as well as $\mathbf{B}$ for the y's corresponding to z_b.

Thus for any consistent candidate active set, we may solve for the corresponding y by solving

$$(I - M_r I_{\mathcal{A}}) y = M_c c_p + M_b b$$

where $I_{\mathcal{A}}$ is a diagonal matrix with $I_{\mathcal{A}(i.i)} = 1$ for $i \in \mathcal{A}$ and $I_{\mathcal{A}(i.i)} = 0$ for $i \in \mathcal{A}^{\mathcal{C}}$

The set of basis variables of the Simplex method correspond to

$$\mathcal{B} = \{i | i \in \mathbf{A} \wedge i \in \mathcal{A}\} \cup \{i | i \in \mathbf{B} \wedge i \in \mathcal{A}^C\}$$

A consistent candidate active set is therefore one for which the number of basis variables equals n_e, and in addition $Q = (I - M_r I_{\mathcal{A}})$ is invertible. For a consistent candidate active set we can calculate the corresponding y from

$$y = Q^{-1}(M_c c_p + M_b b)$$

We then have

$$z_i = y_i, \; \lambda_i = 0 \text{ if } i \in \mathbf{A} \wedge i \in \mathcal{A}$$
$$z_i = 0, \; \lambda_i = -y_i \text{ if } i \in \mathbf{A} \wedge i \in \mathcal{A}^C$$
$$z_i = 0, \; \lambda_i = y_i \text{ if } i \in \mathbf{B} \wedge i \in \mathcal{A}$$
$$z_i = -y_i \; \lambda_i = 0 \text{ if } i \in \mathbf{B} \wedge i \in \mathcal{A}^C$$

Denote by G_B (as above) the matrix consisting of the columns of the matrix G for the variables in $\mathcal{B}$. The inverse of this matrix is required in the Simplex method. The fact that we can extract z directly from the calculated y means that G_B^{-1} can be found from

$$G_B^{-1} = I_{\mathcal{B}} Q_{\mathcal{B}}^{-1} M_b \tag{24}$$

where $I_{\mathcal{B}}$ is a diagonal matrix with $+1$ on the diagonal for row i if $i \in \mathbf{A} \wedge i \in \mathcal{A}$, and -1 on the diagonal of row i if $i \in \mathbf{B} \wedge i \in \mathcal{A}^C$, and $Q_{\mathcal{B}}^{-1}$ consists of the rows i

of Q^{-1} for which $i \in \mathcal{B}$. Furthermore, since the leading n_a rows of M_b are identically zero, we can reduce the computation of the G_B^{-1} to

$$G_B^{-1} = -I_{\mathcal{B}} Q^{-1}_{\mathcal{B},n_a+1:n_a+n_b} G_b^{-1}$$

i.e., involving only the last n_b columns of $Q_{\mathcal{B}}^{-1}$.

Since y contains the Lagrange multipliers, these do not need to be calculated separately, and the identification of the variable to enter the basis is trivial. However, we still need G_B^{-1} to calculate the α_j's required for identifying the variable to leave the active set.

Note, however, that $\alpha = G_B^{-1} G_{(:,i)}$. Instead of calculating the whole of G_B as explained above, we may instead pre-compute $G^I = G_b^{-1} G = \begin{bmatrix} G_b^{-1} G_a & I \end{bmatrix}$, as storing G^I (at least when storing as a full matrix) does not require more memory than storing G. At runtime, we therefore first calculate

$$Q^I_{\mathcal{B}} = -I_{\mathcal{B}} Q^{-1}_{\mathcal{B},n_a+1:n_a+n_b}$$

(i.e., extracting a sub-matrix from Q^{-1}, and flipping the sign of some rows) and then calculate α directly from

$$\alpha = Q^I_{\mathcal{B}} G^I_{(:,i)} \tag{25}$$

where i is the index of the variable entering the basis.

Next, having described how to identify the variable to enter and the variable to leave the basis - and hence what variables enter and/or leave the active set, we need an efficient method for updating Q^{-1}.

Let k be an iteration counter, and at each iteration one variable enters and one variable leaves the basis. We then have

$$Q_{k+1} = Q_k + \begin{bmatrix} M_{4,i} & M_{4,j} \end{bmatrix} \begin{bmatrix} s_i & 0 \\ 0 & s_j \end{bmatrix} \begin{bmatrix} e_i^\top \\ e_j^\top \end{bmatrix} \tag{26}$$

where e_i and e_j are the ith and jth unit vectors, $s_i = 1$ if variable i leaves the active set, and $s_i = -1$ if variable i enters the active set (and similarly for s_j). Clearly, Q_{k+1} is a rank-2 update to Q_k. Knowing Q_k^{-1} we may thus use the Matrix inversion lemma to efficiently calculate Q_{k+1}^{-1}:

$$\begin{aligned} Q_{k+1}^{-1} &= Q_k^{-1} - Q_k^{-1} U (S^{-1} + V Q_k^{-1} U)^{-1} V Q_k^{-1} \\ U &= \begin{bmatrix} M_{4,i} & M_{4,j} \end{bmatrix} \\ S &= \begin{bmatrix} s_i & 0 \\ 0 & s_j \end{bmatrix} \\ V &= \begin{bmatrix} e_i^\top \\ e_j^\top \end{bmatrix} \end{aligned}$$

When calculating Q_{k+1} using the matrix inversion lemma, we make use of the fact that the product VQ_k^{-1} is obtained by simply extracting rows i and j from Q_k^{-1}. Also, if constraint i (or j) is already in the active set, then column i of Q_k is

$$Q_{k(:,i)} = e_i - M_{4(:,i)}$$

and hence

$$Q_k^{-1} M_{4(:,i)} = Q^{-1}_{k(:,i)} - e_i$$

and

$$Q^{-1}_{k(i,:)} M_{4(:,i)} = Q^{-1}_{k(i,i)} - 1$$

Further, noting that $S^{-1} = S$, we calculate Q_{k+1}^{-1} by

1) Extract rows i and j from Q_k^{-1} to obtain VQ_k^{-1}.
2) Multiply VQ_k^{-1} and U (accounting for whether column i or j is already in the active set, as described above)
3) Invert $(S^{-1} + VQ_k^{-1}U)$. This is a 2×2 matrix, so the inversion is trivial.
4) Calculate $Q_k^{-1}U$ (accounting for whether column i or j is already in the active set, as described above)
5) Calculate $Q_k^{-1}U(S^{-1} + VQ_k^{-1}U)^{-1}VQ_k^{-1}$.
6) Add Q_k^{-1} and $Q_k^{-1}U(S^{-1} + VQ_k^{-1}U)^{-1}VQ_k^{-1}$.

C. Initial feasible solution

The Simplex method iteratively improves on a basic feasible solution, and hence an initial basic feasible solution must be available. The standard approach to obtaining a basic feasible solution is known as the Phase 1 of the Simplex method. In Phase 1, the problem description is augmented by slack variables, in such a way that it is trivial to define a basic feasible solution in terms of these slack variables. The objective function for Phase 1 weighs only these slack variables, and a basic feasible solution for the original problem is found if the value of objective function for Phase 1 is zero. However, the Phase 1 procedure can introduce a significant number of slack variables, and can easily find a basic feasible solution far away from the optimal point of the original problem.

Instead, we start with the basic feasible solution from the previous timestep, and follow the approach described by Maros [8] for Phase 1, where the objective function only weighs the negative basis variables. This means that the parameters of the objective function may change for each iteration of the modified Phase 1 procedure. Observe that once the updated Q^{-1} is available, one can extract the rows involved in calculating the basis variables. Having calculated the values of the basis variables, one can then modify the vector of weights for the objective function (based on the value of the basis variables), before updating the Lagrange multipliers. Once a basic feasible solution is found, with no negative basis variables, the objective function weights are fixed to the values in the original LP formulation.

V. NUMERICAL RESULTS

In this section, computational times for solving LP-MPC problems are compared, for the proposed ramp-based LP solver as well as the commercial solver CPLEX [6]. It should be noted that the ramp-based solution is programmed in Matlab m-code, whereas CPLEX is called as a compiled .mex file. The MPC is simulated for 100 timesteps, starting from the same initial state for both solvers. For CPLEX the primal simplex solver is

chosen[2]. Both solvers are warm started from the solution at the previous timestep, for the ramp based solution this is done by specifying the initial basis, whereas for CPLEX the optimal solution from the previous timestep is specified. The two solvers produce the same system responses (as they should), and system responses are therefore not shown. The resulting computation times obtained on a Windows PC are given in Table I. To account for the fact that Windows is not a real time operating system, the average computation time over multiple runs is listed.

Example 1: Consider the double integrator example from [5], [9]. The system dynamics are given by

$$x_{k+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0.3 \end{bmatrix} u_k$$

with the input constrained between -1 and 1, and each state constrained between -5 and 5. The weight on the input is 1 and the weight on each output is 10.

Prediction horizons $N = \{1, 5, 10\}$ are used, to compare the growth in computation times with problem size for the two solvers. The initial state is $x_0 = [5 \ -2]^\top$. ⌟

Example 2: We now study the four-state system from [5], [9]. The discrete-time dynamics are given by

$$x_{k+1} = \begin{bmatrix} 0.928 & 0.002 & -0.003 & -0.004 \\ 0.041 & 0.954 & 0.012 & 0.006 \\ -0.052 & -0.046 & 0.893 & -0.003 \\ -0.069 & 0.051 & 0.032 & 0.935 \end{bmatrix} x_k + \begin{bmatrix} 0 & 0.336 \\ 0.183 & 0.007 \\ 0.090 & -0.009 \\ 0.042 & 0.012 \end{bmatrix} u_k$$

$$y_k = Cx_k = \begin{bmatrix} 0 & 0 & -0.098 & 0.269 \\ 0 & 0 & 0.080 & 0.327 \end{bmatrix} x_k$$

with each input and each measurement constrained to lie between -1 and 1. The weight on each input is 1, and the weight on each output is 10.

The prediction horizon is $N = 30$, and the initial state $x_0 = [25.5724 \ 25.3546 \ 9.7892 \ 0.2448]^\top$. ⌟

[2]It should be noted that this is not necessarily the fastest LP solver available with CPLEX.

TABLE I: Computation times in seconds for simulating LP-MPC for 100 timesteps with CPLEX and the ramp-based solution. Prediction horizon N, number of equality constraints n_e, number of variables n_z.

Example	N	n_e	n_z	CPLEX	Ramp
1	1	8	12	0.4067	0.0200
1	5	40	60	0.4344	0.0306
1	10	80	120	0.4378	0.0504
2	30	300	480	0.6037	2.4898

VI. CONCLUSIONS

A new implementation for the Simplex method for solving LPs, based on ramp functions, is proposed. Initial results indicate computational advantages for problems of small to moderate size. Although significantly reduced computation times may be obtained by programming the ramp-based solution method in a high-performance computer language, results on large problems indicate that significant further improvements in the solution method are required to make it competitive for large problem.

REFERENCES

[1] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, pp. 733–764, 2003.

[2] J. Richalet, A. Raoult, J. L. Testud, and J. Papon, "Model predictive heuristic control: Application to industrial processes," *Automatica*, pp. 413–428, 1978.

[3] R. Cagienard, P. Grieder, E. Kerrigan, and M. Morari, "Move blocking strategies in receding horizon control," *Journal of Process Control*, vol. 17, no. 6, pp. 563–570, 2007.

[4] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *J. of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, December 1998.

[5] G. Valmorbida and M. Hovd, "Quadratic programming with ramp functions and fast online QP-MPC solutions," *Automatica*, vol. 153, p. 111011, 2023.

[6] IBM, "IBM ILOG CPLEX optimizer," Accessed January 2023. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

[7] C. Danielson, "Terminal-cost design for model predictive control with linear stage-costs: A set-theoretic method," *Optimal Control Applications and Methods*, vol. 42, pp. 943–964, 2021.

[8] I. Maros, *Computational Techniques of the Simplex Method*, ser. International Series in Operations Research & Management Science. Springer, 2003.

[9] M. Hovd and F. Stoican, "On the design of exact penalty functions for MPC using mixed integer programming," *Computers & Chemical Engineering*, vol. 70, pp. 104–113, 2014.