A construction-free coordinate-descent augmented-Lagrangian method for embedded linear MPC based on ARX models

Liang Wu\textsuperscript{1}, Alberto Bemporad\textsuperscript{1}

Abstract—This paper proposes an efficient algorithm for solving linear MPC problems based on autoregressive with exogenous terms (ARX) input-output models. Rather than converting the ARX model in state-space form and rely on classical linear MPC methods, we propose an algorithm that directly uses the ARX model to build and solve the MPC problem in a very efficient way. A main feature of the approach is that it completely avoids the online construction step, which makes it very attractive when the ARX model adapted recursively at runtime. The solution algorithm relies on a coordinate-descent augmented Lagrangian (CDAL) method previously proposed by the authors, that we adapt here to exploit the special structure of ARX-based MPC. The implementation of the resulting CDAL-ARX algorithm is matrix-free and library-free, and hence amenable for deployment in industrial embedded platforms. We show the efficiency of CDAL-ARX in two numerical examples, also in comparison with MPC implementations based on state-space models and general-purpose quadratic programming solvers.

Index Terms—ARX, State-Space, Model Predictive Control, Construction-free

I. INTRODUCTION

Model Predictive Control (MPC) is an advanced technique to control multi-input multi-output systems subject to constraints, and its core idea to predict the evolution of the controlled system by means of a dynamical model, solve an optimization problem over a finite time horizon, only implement the control input at the current time, and then repeat the optimization again at the next sample [1]. In earlier MPC developments, some methods shared the same receding horizon control idea, under different names. The Model Predictive Heuristic Control (MPHC) [2], the Model Algorithmic Control (MAC) [3] used a finite impulse response model, the Dynamic Matrix Control (DMC) employed a truncated step-response model [4], and the Generalized Predictive Control (GPC) involved a transfer function model [5]. As the MPC field has grown, state-space (SS) models replaced input-output (I/O) models, and most MPC theory is based on SS formulations [6].

However, in industrial control applications, MPC based on input-output models, such as the autoregressive model with exogenous terms (ARX) model, may still be preferable [1], for two main reasons: (1) there is no need of a state-observer; (2) I/O models are easier to identify and to adapt online (such as using recursive least-squares algorithms), which makes them widely used in adaptive control [7]. In particular, the latter is particularly appealing in practical cases in which the dynamics of the systems changes during operations, such as in the case of changes of mass and inertia in rockets due to fuel consumption, wear of heating equipment in chemical processes, and many others. In such a linear time-varying setting, the online computation time is due to both constructing and solving the optimization problem associated with MPC; indeed, often constructing and solving the MPC problem have comparable costs, especially when warm-starting strategies are employed and set-points change slowly. The online construction of the MPC problem is also required in the case of linear parameter varying ARX (LPV-ARX) models, in which model parameters depend on a measured time-varying signal, the so-called scheduling variable. Control methods based on LPV descriptions have been widely used in a wide range of applications, in particular to handle nonlinear dynamics [8]. Besides numerical complexity, in fast control applications often matrix calculus cannot be implemented in cheap embedded platforms by relying on complex linear algebra libraries.

A. Contribution

In this paper, we focus on MPC problems based on ARX models (ARX-based MPC), due to their ease of online identification, and address the above requirements by proposing an adaptation of the coordinate descent augmented Lagrangian (CDAL) method we recently introduced in [9] for MPC based on state-space models. The resulting CDAL-ARX approach inherits the main benefits of CDAL, namely its construction-free, matrix-free, and library-free features. Although most implementations of first-order algorithms can also be matrix-free and library-free, usually they are not construction-free; for example, the OSQP solver [10], which is based on the alternating direction method of multipliers (ADMM), needs to cache a factorization of the Hessian matrix of the quadratic costs, which is different for different problems, and the proportional–integral projected gradient method [11] also requires calculating the largest singular value of the equality constraints matrix for determining its step-size parameters.

Most ARX-based MPC algorithms in the literature first convert the ARX model into SS form, treat the problem as a standard SS-based MPC problem [12], and then construct and solve a condensed or sparse quadratic programming (QP) problem. In [13], the I/O model-based MPC problem is formulated as a bounded variable least-squares (BVLS) problem without resorting to ARX-to-SS transformations. The method relies on relaxing the dynamic equality constraints by introducing a (usually large) penalty parameter, therefore possibly introducing intrinsic modeling errors and numerical
difficulties, that CDAL-ARX does not suffer.

II. ARX-BASED MPC PROBLEM FORMULATION

Consider the multi-input multi-output (MIMO) ARX model described by
\[
y_t = \sum_{i=1}^{n_a} A(i)y_{t-i} + \sum_{i=1}^{n_b} B(i)u_{t-i} \tag{1}
\]
where \( y_t \in \mathbb{R}^{n_y} \) and \( u_t \in \mathbb{R}^{n_u} \) are the output and input of the system, respectively, \( A(i) \in \mathbb{R}^{n_y \times n_y}, \) \( i = 1, \ldots, n_a, \) and \( B(i) \in \mathbb{R}^{n_y \times n_u}, \) \( i = 1, \ldots, n_b, \) define the model order of the ARX model.

This paper considers the following MPC tracking formulation based the ARX model (1):
\[
\min_{Y,U,\Delta U} \frac{1}{2} \sum_{t=1}^{T} \| (y_t - r_t) \|^2_W + \| \Delta u_{t-1} \|^2_W \tag{2}
\]
s.t.
\[
y_t = \sum_{i=1}^{n_a} A(i)y_{t-i} + \sum_{i=1}^{n_b} B(i)u_{t-i}, \quad t = 1, \ldots, T
\]
\[
\Delta u_t = u_t - u_{t-1}, \quad t = 0, \ldots, T - 1
\]
\[
y_{\min} \leq y_t \leq y_{\max}, \quad t = 1, \ldots, T
\]
\[
u_{\min} \leq u_t \leq u_{\max}, \quad t = 0, \ldots, T - 1
\]
\[
\Delta u_{\min} \leq \Delta u_t \leq \Delta u_{\max}, t = 0, \ldots, T - 1
\]
where \( T \) is the prediction horizon, \( W_y \geq 0 \) and \( W_{\Delta u} \geq 0 \) are positive semi-definite diagonal matrices on the outputs and the input increments, respectively, \( r_t, t = 1, \ldots, T \) are the future desired set-point vectors, \( \Delta u_{t-1} \) are the input increments, \( [y_{\min}, y_{\max}], \) \([u_{\min}, u_{\max}], \) and \([\Delta u_{\min}, \Delta u_{\max}] \) define box constraints on outputs, inputs, and input increments, respectively, and \( Y = (y_1, \ldots, y_T), \) \( U = (u_0, \ldots, u_{T-1}) \), and \( \Delta U = (\Delta u_0, \ldots, \Delta u_{T-1}) \) are the optimization variables.

III. COORDINATE DESCENT AUGMENTED LAGRANGIAN METHOD

In [9], we proposed a coordinate-descent augmented Lagrangian (CDAL) method for SS-based MPC problems. We want to adapt here the method to solve problem (2) without computing a state-space realization of the ARX model (1), while retaining the construction-free, matrix-free, and library-free properties of CDAL.

A. Augmented Lagrangian method

The following assumptions are needed to ensure the convergence of the Augmented Lagrangian method.

Assumption 1: Problem (2) has a feasible solution.

Note that Assumption 1 is satisfied in all practical situations in which the reference \( r_t \) is far enough from the output bounds and the prediction horizon \( T \) is long enough.

Assumption 2: The equality constraint matrix arising from stacking all the equality constraints in (2) is full rank at the optimal solution of the problem.

Let \( \mathcal{Y}, \mathcal{U}, \) and \( \Delta \mathcal{U} \) denote the hyper-boxes on \( Y, \ U, \) and \( \Delta U, \) respectively, defined by the box constraints in (2), respectively. The bound-constrained Augmented Lagrangian function \( \mathcal{L}_{\rho} : \mathcal{Y} \times \mathcal{U} \times \Delta \mathcal{U} \times \mathbb{R}^{T n_y} \times \mathbb{R}^{T n_u} \to \mathbb{R} \) is given by
\[
\mathcal{L}_{\rho}(Y, U, \Delta U, \Lambda, \Gamma) = \frac{1}{2} \sum_{t=1}^{T} \| (y_t - r_t) \|^2_W + \| \Delta u_{t-1} \|^2_W \tag{3}
\]
\[
+ \sum_{t=1}^{T} \chi_t^* \left( \sum_{i=1}^{n_a} A(i)y_{t-i} + \sum_{i=1}^{n_b} B(i)u_{t-i} - y_t \right) - \sum_{t=1}^{T} \gamma_t \left( u_{t-2} + \Delta u_{t-1} - u_{t-1} \right)
\]
\[
+ \frac{\rho}{2} \sum_{t=1}^{T} \| u_{t-2} + \Delta u_{t-1} - u_{t-1} \|^2_2
\]
where \( \Lambda = \{ \lambda_t \} \) and \( \Gamma = \{ \gamma_t \}, \forall t = 1, \ldots, T \) are the dual vectors associated to the equality constraints induced by the ARX model and the input increments, respectively, and \( \rho \) is the penalty parameter. According to [14], the scaled AL method (ALM) iterates the following updates
\[
(y^k, u^k, \Delta u^k) = \arg \min_{Y, U, \Delta U} \mathcal{L}_{\rho}(Y, U, \Delta U, \Lambda^{k-1}, \Gamma^{k-1}) \tag{4a}
\]
\[
\lambda_t^k = \lambda_t^{k-1} + \sum_{i=1}^{n_a} A(i)y_{t-i}^k + \sum_{i=1}^{n_b} B(i)u_{t-i}^k - y_t^k \tag{4b}
\]
\[
\gamma_t^k = \gamma_t^{k-1} + u_{t-2}^k + \Delta u_{t-1}^k - u_{t-1}^k, \forall t = 1, \ldots, T \tag{4c}
\]
The minimization step (4a) updates the primal vector, Steps (4b) and (4c) update the dual vectors. We refer the reader to [14] for a well-known convergence proof of ALM under Assumptions 1. To improve the speed of convergence of ALM, [15] proposed an accelerated version of ALM whose convergence rate is \( O(1/k^2) \) for linearly constrained convex programs by using Nesterov’s acceleration technique [16]. The accelerated ALM algorithm for MPC problems has been summarized in our previous work [9].

B. Coordinate-descent method

Sub-problem (4a) is a strongly convex box-constrained QP problem, which can be solved by many methods. Among others, as we showed in [9], problem (4a) can be solved by a simple coordinate-descent method, which minimizes the objective function along only one coordinate direction at each iteration while keeping the other coordinates fixed [17]. A convergence proof of at-least linear convergence when solving convex differentiable minimization problems was shown in [17]. Under Assumption 1 (non-emptiness of the feasible set) and since the objective function \( \mathcal{L}_{\rho}(\cdot) \) is continuously differentiable and convex with respect to each coordinate, the CD method proceeds repeatedly for \( k = \ldots, T \).
1, 2, . . . , as follows:

\[ \frac{1}{z_{jk}} = \arg\min_{z_{jk} \in \mathbb{Z}} \mathcal{L}_\rho(z_{jk}, z_{jk}^{-1}, \Lambda^{-1}, \Gamma^{-1}) \]  

(5b)

where \( z = \left[ y_1, u_0, \Delta u_0, \ldots, y_T, u_T, \Delta u_T \right]' \) is the optimization vector, \( z \in \mathbb{Z} \triangleq \mathbb{Y} \times \mathbb{U} \times \Delta \mathbb{U}, \mathbb{Z} \subseteq \mathbb{R}^{n_z}, n_z \triangleq |T| (n_y + n_u + n_u) \). We denote by \( \mathcal{L}_\rho \) the value of \( \mathcal{L}_\rho(z_{jk}, z_{jk}^{-1}, \Lambda^{-1}, \Gamma^{-1}) \) when \( z_{jk} = z_{jk}^{-1} \) is fixed. Here \( z_{jk}^{-1} \) denotes the subvector obtained from \( z \) by eliminating its \( j_k \)th component \( z_{jk} \). The convergence of the iterations (5) depends on the coordinate picking rule, namely how \( j_k \) is chosen. Existing research works have analyzed the influence of different coordinate selection rules such as the cyclic rule and the random selection rule, on the convergence rate of the coordinate descent method. We choose the simplest variant using cyclic coordinate search to favor implementation simplicity. In fact, the cyclic implementation preserves the order of optimization variables with respect to the prediction horizon \( t \), type (output \( y_t \), input \( u_t \), or input increment \( \Delta u_t \)) and component, so that reconstructing the coordinate that is currently optimized in is immediate. The implementation of one pass through all \( n_z \) coordinates using cyclic CD is reported in Procedure 1. In Procedure 1, the operator \( \{s, \sigma\} = \text{CCD}(M, d, \sigma) \) represents one pass of iteration of the reverse cyclic CD method through all its \( n_s \) coordinates \( s_1, \ldots, s_{n_s} \) for the box-constrained QP \( \min_{\min c \in [s, \sigma]} s'Ms + s'd \) that is to execute the following \( n_s \) iterations

\[
\text{for } i = 1, \ldots, n_s \ni \begin{align*}
\hat{s}_i &\leftarrow \max(\hat{s}_i, \min(s_i, s_i - \frac{1}{M_{i,i}} (M_{i,i} s + d_i))) \\
\sigma &\leftarrow \sigma + (\hat{s}_i - s_i)^2 \\
s_i &\leftarrow \hat{s}_i 
\end{align*} 
\]

(6)

The definitions of the quantities \( c_t, f_i, y_t \) (\( t = 1, 2, \ldots, T \)) used in Procedure 1 are reported in Appendix I which shows that they involve several matrix-vectors multiplications. To eliminate their explicit calculation, we propose here below an efficient coupling scheme between CD and AL that reduces the cost per iteration, without changing the rate of convergence of the algorithm.

C. Efficient coupling scheme between CD and AL

Our proposed efficient coupling scheme exploits the fact that CD only updates one coordinate each time, and the execution of the operator \( \text{CCD}(\cdot) \) involves the next update of dual Lagrangian vectors. Here we take Step 2.2 of Procedure 1 as an example, which has been modified from equation (6) to Procedure 2. Note that the dual Lagrangian vectors used in Procedure 2 have been updated before Procedure 2. The symbols \( \{D^\rho_{jj}, D^\rho_{jj'}, \ldots, D^\rho_{T-1,j-1}\} \) denote the diagonal elements of their Hessian matrices used in Step 2.2.

Procedure 1 Full pass of cyclic coordinate descent on all block variables

\textbf{Input}: \( \Lambda = \{\lambda_1, \ldots, \lambda_T\}, \Gamma = \{\gamma_1, \ldots, \gamma_T\}, Y = \{y_1, \ldots, y_T\}, U = \{u_0, \ldots, u_{T-1}\}, \Delta U = \{\Delta u_0, \ldots, \Delta u_{T-1}\} \); MPC settings \( A(1), \ldots, A(n_a), B(1), \ldots, B(n_b), W^y, \Delta u, y_{\min}, y_{\max}, u_{\min}, u_{\max}, \Delta u_{\min}, \Delta u_{\max} \); parameter \( \sigma, \rho > 0 \).

1. \( \sigma \leftarrow 0 \);
2. \( \text{for } t = 1, \ldots, T - 1 \)
   2.1. \( j = \min(n_a, T - t); \)
   2.2. \( \{y_t, \sigma\} \leftarrow \text{CCD}(\frac{1}{\rho W^y + I} + \sum_{i=1}^{T} A(i)'A(i), e_t, \sigma\}_{\gamma_{\max}}^{\gamma_{\min}}; \)
   2.3. \( j = \min(n_b, T - t + 1); \)
   2.4. \( \{u_{t-1}, \sigma\} \leftarrow \text{CCD}(2I + \sum_{i=1}^{T} \sigma B(i)'B(i), f_t, \sigma\}_{\gamma_{\max}}^{\gamma_{\min}}; \)
   2.5. \( \{\Delta u_t, \sigma\} \leftarrow \text{CCD}(\frac{1}{\rho W^y\Delta u + I} + \sum_{i=1}^{T} A(i)'A(i), e_t, \sigma\}_{\gamma_{\max}}^{\gamma_{\min}}; \)
3. \( \{y_T, \sigma\} \leftarrow \text{CCD}(\frac{1}{\rho W^y + I} + \sum_{i=1}^{T} A(i)'A(i), e_T, \sigma\}_{\gamma_{\max}}^{\gamma_{\min}}; \)
4. \( \{u_{T-1}, \sigma\} \leftarrow \text{CCD}(\Delta u_t, \sigma\}_{\gamma_{\max}}^{\gamma_{\min}}; \)
5. \( \{\Delta u_{T-1}, \sigma\} \leftarrow \text{CCD}(\frac{1}{\rho W^y + I} + \sum_{i=1}^{T} A(i)'A(i), e_T, \sigma\}_{\gamma_{\max}}^{\gamma_{\min}}; \)
6. \textbf{end.}

\textbf{Output}: \( Y, U, \Delta U, A, \Gamma, \sigma. \)

Procedure 2 One pass of cyclic coordinate descent for Step 2.2 of Procedure 1 after using efficient coupling scheme

\textbf{Input}: \( j = \min(n_a, T); \)
1. \( \sigma \leftarrow 0; \)
2. \( \text{for } i = 1, \ldots, n_y \)
   1.1. \( s \leftarrow -\lambda_{t,i} + \sum_{j=1}^{T} A(n_i)'A(i); \)
   1.2. \( \theta \leftarrow \left[ y_t,i - \frac{1}{\rho W^y(y_t,i - \tau_{t,i}) + \sigma} \right]_{\gamma_{\max}}^{\gamma_{\min}}; \)
   1.3. \( \Delta \leftarrow \theta - y_t,i; \)
   1.4. \( \sigma \leftarrow \sigma + \Delta; \)
   1.5. \( y_t,i \leftarrow \theta; \)
   1.6. \( \lambda_{t,i} \leftarrow \lambda_{t,i} + \Delta; \)
   1.7. \text{for } n_i = 1, \ldots, j \text{ do} \)
   1.7.1. \( \lambda_{t+n_i,i} \leftarrow \lambda_{t+n_i,i} + \Delta \cdot A(n_i); \)
2. \textbf{end.}

\textbf{Output}: \( y_t, \lambda_{t}, \lambda_{t+1}, \ldots, \lambda_{t+j}, \sigma. \)

and

\[
\text{for } t = 1, \ldots, T - 1 \ni j = \min(n_a, T - t); \quad D^\rho_{jj} \leftarrow \text{diag} \left( \frac{1}{\rho W^y + I} + \sum_{i=1}^{T} A(i)'A(i) \right) \]
\textbf{end}
\[
D^\rho_{T-1,j} \leftarrow \frac{1}{\rho W^y + I} 
\]

To avoid repeating division operations, the values \( \{\frac{1}{\rho W^y}, \frac{1}{\rho W^y + I}, \ldots, \frac{1}{\rho W^y + I} \} \) are cached before the iterations start. The other steps involving the operator \( \text{CCD}(\cdot) \) in Procedure 1 follow the same idea.
D. Algorithm

Summarizing all the ingredients described in the previous sections, we obtain the construction-free ARX-based MPC Algorithm\textsuperscript{3} which we call CDAL-ARX. Here, construction-free means that CDAL-ARX directly uses the ARX model coefficients without the need of constructing a QP problem explicitly. Note that the main update of the Lagrangian variables in Algorithm\textsuperscript{3} is placed early in Step 2.1 which is different from the original version of Algorithm 1 in [9] because the CD method allows the use of our proposed efficient coupling scheme. The quantities $N_{\text{out}}$ and $N_{\text{in}}$ denote the maximum number of AL (outer-loop) and CD (inner-loop) iterations, respectively. The tolerances $\epsilon_{\text{out}}$ and $\epsilon_{\text{in}}$ define the stopping criteria of the outer and inner iterations, respectively. We remark that Algorithm\textsuperscript{3} is matrix-free and we could implement it in 150 lines of C code without using any external library.

Algorithm 3 Accelerated cyclic CDAL algorithm for ARX-based MPC

Input: primal/dual warm-start $Y = \{y_1, y_2, \ldots, y_T\}$, $U = \{u_0, u_1, \ldots, u_{T-1}\}$, $\Delta U = \{\Delta u_0, \Delta u_1, \ldots, \Delta u_{T-1}\}$, $\Lambda^0 = \{\lambda_1, \lambda_2, \ldots, \lambda_T\}$, $\Gamma^0 = \{\gamma_1, \gamma_2, \ldots, \gamma_T\}$; History inputs and output data $\{y_0, y_{-1}, \ldots, y_{1-n_a}\}$, $\{u_{-1}, u_{-2}, \ldots, u_{1-n_b}\}$; MPC settings $\{A(1), A(2), \ldots, A(n_a), B(1), B(2), \ldots, B(n_b), W^y, W^u, \Delta u_{\text{min}}, y_{\text{min}}, u_{\text{min}}, \Delta u_{\text{max}}, \Delta u_{\text{max}}\}$; Algorithm settings $\{\rho, N_{\text{out}}, N_{\text{in}}, \epsilon_{\text{out}}, \epsilon_{\text{in}}\}$

1. $\alpha_1 \leftarrow 1; \hat{\Lambda}^0 \leftarrow \Lambda^0; \hat{\Gamma}^0 \leftarrow \Gamma^0$;
2. for $k = 1, 2, \ldots, N_{\text{out}}$ do
   2.1. for $t = 1, 2, \ldots, T$ do
      2.1.1. $\lambda_i^k = \hat{\Lambda}_i^k - 1 + (\sum_{i=1}^{n_a} A(i) y_{t-i} + \sum_{j=1}^{n_b} B(i) u_{t-j} - y_t)$
      2.1.2. $\gamma_i^k = \hat{\Gamma}_i^k - 1 + (u_{t-1} + \Delta u_{t-1} - u_{t-1})$;
   2.2. for $k_{\text{in}} = 1, 2, \ldots, N_{\text{in}}$ do
      2.2.1. $(Y, U, \Delta U, \sigma)$ ← Procedure 2 with use of Procedure 2
      2.2.2. if $\sigma \leq \epsilon_{\text{in}}$ break the loop;
   2.3. if $\|\Lambda^k - \hat{\Lambda}^{k-1}\|^2 < \epsilon_{\text{out}}$ stop;
   2.4. $\alpha_{k+1} \leftarrow \frac{1}{1 + \alpha_k}$
   2.5. $\hat{\Lambda}^k \leftarrow \frac{\alpha_k}{\alpha_{k+1}}(\hat{\Lambda}^k - \hat{\Lambda}^{k-1})$;
3. end.

Output: $Y, U, \Delta U, \Lambda, \Gamma$

IV. Numerical examples

In this section, we test our proposed ARX-based MPC algorithm against other more traditional MPC approaches. By using the transformation between state-space based MPC and ARX-based MPC reported in Appendix 1 we can obtain efficient implementations of MPC-to-QP condensed or sparse constructions. The best choice between condensed and sparse QP forms mainly depends on the number of states $n_x$, control inputs $n_u$, and the length of the prediction horizon $T$ [18]. For numerical comparisons with our ARX-based MPC algorithm, this paper considers both a condensed and a sparse QP construction and uses qpOASES [19] to solve the former form, and OSQP [10] to solve the latter. The reported comparison simulation results were obtained on a MacBook Pro with a 2.7 GHz 4-core Intel Core i7 and 16GB RAM. Algorithm\textsuperscript{3} qpOASES v3.2 and OSQP v0.6.2 are all executed in MATLAB R2020a via their C-mex implementations.

A. Problem descriptions

1) Time-varying ARX model example: one notable feature of ARX models is their ease to be updated at runtime, which makes them particularly appealing when the system dynamics cannot be well captured by a single linear time-invariant model. Our CDAL-ARX algorithm can take advantage of its construction-free feature to avoid the computation cost of the online construction step. We tested CDAL-ARX on randomly-generated two-input-two-output ARX models with order $n_a = 4$ and $n_b = 4$ and time-varying system matrices. For demonstration purposes, here below we report one instance of them, whose ARX coefficient matrices $A(1), \ldots, A(4), B^1(1), \ldots, B^3(4)$ at time $t$ are given by

\begin{align*}
A(i) & = A(i) + 0.1 M_t, i = 1, \ldots, 4 \\
B(i) & = B(i) + 0.1 M_t, i = 1, \ldots, 4
\end{align*}

where $A(1) = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}, A(2) = \begin{bmatrix} 0.7 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}, A(3) = \begin{bmatrix} 0.5 & 0.1 \\ 0.1 & 0.5 \end{bmatrix}, A(4) = \begin{bmatrix} 0.3 & 0.1 \\ 0.1 & 0.3 \end{bmatrix}$, $B(1) = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, B(2) = \begin{bmatrix} 0.8 & 0.4 \\ 0.4 & 0.8 \end{bmatrix}, B(3) = \begin{bmatrix} 0.6 & 0.3 \\ 0.3 & 0.6 \end{bmatrix}$, $B(4) = \begin{bmatrix} 0.4 & 0.2 \\ 0.2 & 0.4 \end{bmatrix}$, $M_t = \begin{bmatrix} \sin(\phi) & \cos(\phi) \\ \cos(\phi) & \sin(\phi) \end{bmatrix}$.

2) DNN-based LPV-ARX model example: We tested on CDAL-ARX on randomly-generated two-input-two-output quasi-LPV-ARX models of larger order $n_a = 6$ and $n_b = 6$, whose coefficient matrices are piecewise affine (PWA) maps of the scheduling vector $w_{t-1}$

\begin{align*}
\begin{bmatrix} y_t(1) \\ y_t(2) \end{bmatrix} & = \begin{bmatrix} N_1(w_{t-1})y_{t-1} \\ N_2(w_{t-1})y_{t-1} \end{bmatrix} \\
where \quad w_{t-1} & = \begin{bmatrix} y_{t-1,1}, \ldots, y_{t-1,6}, u_{t-1,1}, \ldots, u_{t-1,6} \end{bmatrix} \in \mathbb{R}^{24}, \quad w_{t-1} \in \mathbb{R}^{22}, \quad N_1, N_2 \in \mathbb{R}^{22 \times 24}
\end{align*}

are deep feedforward neural networks with three layers and ReLU activation function, namely $N_1(w_{t-1}) = W_{1,1} \max(0, W_{1,2} \max(0, W_{1,3} v_{t-1} + b_{1,1}) + b_{1,2}) + b_{1,3}, N_2(w_{t-1}) = W_{2,1} \max(0, W_{2,2} \max(0, W_{2,3} v_{t-1} + b_{2,1}) + b_{2,2}) + b_{2,3}$. Here we choose the number of neurons in each hidden layer as three times the number of inputs according to [20], that is, $W_{1,1} \in \mathbb{R}^{66 \times 22}, b_{1,1}$ and $b_{2,1} \in \mathbb{R}^{66}, W_{1,2} \in \mathbb{R}^{66 \times 66}, b_{1,2}$ and $b_{2,2} \in \mathbb{R}^{66}, W_{1,3} \in \mathbb{R}^{66 \times 66}, b_{1,3}$ and $b_{2,3} \in \mathbb{R}^{24}$. For demonstration purposes, we
define $b_{1,3}, b_{2,3}$ by collecting the coefficients defining $A(1), \ldots, A(4), A(4), B(1), \ldots, B(4), B(4)$ as in [8]. The remaining network parameters are randomly generated uniformly between 0 and 0.1.

At each time $t$, the linear model consumed by our CDAL-ARX algorithm is given by evaluating the deep ReLU networks as in (9).

In both examples, we use the same MPC parameters $W_y = I$, $W_u = 0.1I$, $\begin{bmatrix} [y_{\text{min}}, y_{\text{max}}] & [-1,1], \ [u_{\text{min}}, u_{\text{max}}] & [-1,1], \ [\Delta u_{\text{min}}, \Delta u_{\text{max}}] & [-1,1]. \end{bmatrix}$ Different prediction horizon lengths $T$ are used to assess numerical performance, namely $T = 10, 20, 30$. The initial conditions are $y_{-3} = y_{-2} = y_{-1} = y_0 = [0 0]^T$, and $u_{-3} = u_{-2} = u_{-1} = [0 0]^T$. In the two examples, the closed-loop simulation is run over 200 sampling steps, and the desired references for $y_1$ and $y_2$ are randomly changed every 20 steps. Warm-start used in all solvers (qpOASES, OSQP, CDAL-ARX). We keep default solver settings in both qpOASES and OSQP, so that they produce solutions of similar precision, that is measured in terms of Euclidean distance (since qpOASES belongs to the class of active-set methods, in principle it always provides a high-precision solution at termination, so its solution quality cannot be tuned as easily as in the case of ADMM). For a fair comparison, in the two examples we set $\epsilon_{\text{in}} = 10^{-6}$ and $\epsilon_{\text{out}} = 10^{-6}$ under $\rho = 1$ to define the stopping criteria of our CDAL-ARX solver, so to obtain closed-loop control sequences with similar precision. In both examples, the generated closed-loop simulation results are almost indistinguishable, see Figures [1(a)] and [1(b)] respectively, which show good tracking performance and no violation in input and output constraints.

Using the qpOASES and OSQP solvers require the online construction of the QP problem, whose computation time must be counted in the total time. Table I lists the solution time of CDAL-ARX and lists the construction and solution time when using qpOASES (condensed construction) and OSQP (sparse construction). From Table I it can be noticed that CDAL-ARX is always solving the MPC problem in a smaller CPU time, when compared to the sum of construction and solution time of qpOASES and OSQP. Moreover, as the prediction horizon increases, qpOASES and OSQP may fail to solve the problem due to the ill-conditioning issue. Note also that the computation time of CDAL-ARX is often shorter than the pure solution time of qpOASES and OSQP (i.e., not counting the construction time), which seems to indicate that the reported speed-ups are due to both adopting the proposed augmented Lagrangian method and avoiding the use of state-space models.

V. CONCLUSION

This paper has introduced a solution algorithm for solving MPC problems based on ARX models that avoids constructing the associated QP problem explicitly. Due to its matrix-free and library-free features, the proposed CDAL-ARX algorithm can be useful in adaptive embedded linear MPC applications based on ARX models, especially when combined with a fast and robust recursive linear identification method. Future research will address extending the method to handle soft output constraints, so to relax Assumption I.

TABLE I

| Examples               | T       | CDAL-ARX avg, max | qpOASES avg, max | OSQP avg, max |
|------------------------|---------|-------------------|-----------------|--------------|
| Time-varying ARX       | 10      | 0.14, 1.5         | 0.42, 2.8†       | 0.41, 2.9†   |
|                        | 20      | 0.25, 2.8         | 0.08, 1.4†       | 0.18, 0.92†  |
|                        | 30      | 0.36, 3.6         | 1.2, 6.3†       | 1.0, 3.8†    |
| DNN-based LPV-ARX      | 10      | 0.51, 2.5         | 0.46, 3.2†       | 0.42, 3.6†   |
|                        | 20      | 1.2, 4.6          | 0.57, 3.9†       | 1.1, 14†    |
|                        | 30      | 2.0, 7.3          | 1.2, 5.5†       | 0.97, 4.5†   |
|                        |         | fail              | 16†, 32†         |              |

* construction time, † solution time. For qpOASES and OSQP the time to evaluate the MPC law is the sum of construction and solution time.

REFERENCES

[1] S.J. Qin and T.A. Badgwell. A survey of industrial model predictive control technology. Control engineering practice, 11(7):733–764, 2003.
[2] J. Richealet, A. Rault, J.L. Testud, and J. Papin. Model predictive heuristic control: Applications to industrial processes. Automatica, 14(5):413–428, 1978.
[3] R. Ramine and K.M. Raman. Model algorithmic control (MAC); basic theoretical properties. Automatica, 18(4):401–414, 1982.
[4] C.R. Cutler and B.L. Ramaker. Dynamic matrix control-A computer control algorithm. In joint automatic control conference, page 72, 1980.
[5] D.W. Clarke, C. Mohtadi, and P.S. Tuffs. Generalized Predictive Control—Part I. The basic algorithm. Automatica, 23(2):137–148, 1987.
[6] D.Q. Mayne. Model predictive control: Recent developments and future promise. Automatica, 50(12):2967–2986, 2014.
[7] K.J. Aström and B. Wittenmark. Adaptive control. Courier Corporation, 2013.
[8] R. Toth. Modeling and identification of linear parameter-varying systems, volume 403. Springer, 2010.
[9] L. Wu and A. Bemporad. A Simple and Fast Coordinate-Descent Augmented-Lagrangian solver for Model Predictive Control. arXiv preprint arXiv:2109.10205, 2021.
[10] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. Mathematical Programming Computation, 12(4):637–672, 2020.
[11] Y. Yu, P. Elang, U. Topcu, and B. Ackernye. Proportional–integral projected gradient method for conic optimization. Automatica, 142:110359, 2022.
[12] J.K. Huous, N.K. Poulson, S.B. Jørgensen, and J.B. Jørgensen. ARX-model based model predictive control with offset-free tracking. In Computer Aided Chemical Engineering, volume 28, pages 601–606. Elsevier, 2010.
[13] N. Saraf and A. Bemporad. Fast model predictive control based on linear input/output models and bounded-variable least squares. In 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pages 1919–1924. IEEE, 2017.
[14] D.P. Bertsekas. Constrained optimization and Lagrange multiplier methods. Academic press, 2014.
[15] M. Kang, M. Kang, and M. Jung. Inexact accelerated augmented Lagrangian methods. Computational Optimization and Applications, 62(2):373–404, 2015.
[16] Y.E. Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. In Dokl. akad. nauk Ssr, volume 269, pages 543–547, 1983.
APPENDIX I

QUANTITIES USED IN PROCEDURE [10]

The quantities $e_t, f_t, g_t$ used in Procedure [10] are defined for $t = 1, 2, \cdots, T$ as follows:

\[ e_t = -W y_T^T - (\lambda_T + \sum_{i=1}^{n_u} A(i) y_{t-i}) + \min_{(n_u,T-t)} \left( \sum_{i=1}^{n_u} A(n_u) (\lambda_T + \sum_{i=1}^{n_u} A(i) y_{t-i}) \right) \]

\[ + \sum_{n_i=1}^{n_u} A(n_i) (\lambda_{t+n_i} + \sum_{i=1}^{n_u} A(i) y_{t+n_i}) \]

\[ + \sum_{i=1}^{n_u} B(i) u_{t+n_i-1} - y_{t+n_i} \]

\[ f_t = -(\gamma_T + u_{t-2} + \Delta u_{t-1}) + (\gamma_{t+1} + \Delta u_T + u_T) \]

\[ + \sum_{n_i=1}^{n_u} B(n_i) (\lambda_{t+n_i} + \sum_{i=1}^{n_u} A(i) y_{t+n_i}) \]

\[ + \sum_{i=1}^{n_u} B(i) u_{t+n_i-1} - y_{t+n_i} \]

\[ g_t = \gamma_T + u_{t-2} - u_{t-1} \]

\[ e_T = -W y_T^T - (\lambda_T + \sum_{i=1}^{n_u} A(i) y_{T-i}) + \min_{(n_u,T-t)} \left( \sum_{i=1}^{n_u} A(n_u) (\lambda_T + \sum_{i=1}^{n_u} A(i) y_{T-i}) \right) \]

\[ + \sum_{i=1}^{n_u} A(i) y_{T-i} + \sum_{i=1}^{n_u} B(i) u_{T-i} - y_T \]

\[ f_T = -(\gamma_T + u_{T-2} + \Delta u_{T-1}) + B(1) (\lambda_T) \]

\[ + \sum_{i=1}^{n_u} A(i) y_{T-i} + \sum_{i=1}^{n_u} B(i) u_{T-i} - y_T \]

\[ g_T = \gamma_T + u_{T-2} - u_{T-1} \]

Fig. 1. Closed-loop tracking results

APPENDIX II

ARX-MPC TO SS-MPC CONVERSION

The above ARX-based MPC problem (2) can be converted to the following SS-based MPC problem

\[
\min_{X, \Delta U} \frac{1}{2} \sum_{t=1}^{T} \left( \| (C y x_t - r_t) \|_{W_y}^2 + \| \Delta u_{t-1} \|_{W_u}^2 \right)
\]

s.t.

\[ x_{t+1} = A_c x_t + B_c \Delta u_t, t = 0, \ldots, T - 1 \]

\[ y_{t+1} \leq C y x_t, t = 0, \ldots, T - 1 \]

\[ u_{t+1} \leq C u x_t, t = 0, \ldots, T - 1 \]

\[ \Delta u_{min} \leq \Delta u_t \leq \Delta u_{max}, t = 0, \ldots, T - 1 \]

\[ x_0 = [y_{T-1}^T, \ldots, y_0^T, u_{T-1}^T, \ldots, u_0^T]^T \]

where $X = (x_1, \ldots, x_T)$, $\Delta U = (\Delta u_0, \ldots, \Delta u_{T-1})$, $A_c \in \mathbb{R}^{n_{ac} \times n_{ac}}$, $n_{ac} = n_x n_y + (n_b - 1) n_u$, $B_c \in \mathbb{R}^{n_{ac} \times n_u}$, $C_y \in \mathbb{R}^{n_{ac} \times n_x}$ and $C_u \in \mathbb{R}^{n_{ac} \times n_u}$ are defined as

\[
A_c = \begin{bmatrix}
0 & I & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & I & 0 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
A_{n_1} & A_1 & B_1 & B_1 & B_1 & B_1 & B_1 & B_1 \\
\end{bmatrix}
\]

\[
B_c = \begin{bmatrix}
0 & 0 & \cdots & 0 & I & 0 & \cdots & 0 \\
B_1 & 0 & \cdots & 0 & 0 & I & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & I \\
\end{bmatrix}
\]

\[
C_y = \begin{bmatrix}
0 & 0 & \cdots & 0 & I & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & I & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & I \\
\end{bmatrix}
\]

\[
C_u = \begin{bmatrix}
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & I \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & I \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & I \\
\end{bmatrix}
\]