Block Factorization of Step Response Model Predictive Control Problems

D. K. M. Kufoalor[a], G. Frison[c], L. Imsland[a], T. A. Johansen[a,b], J. B. Jørgensen[c]

[a] *Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), O.S. Bragstads plass 2D N-7491 Trondheim, Norway.*
[b] *Center for Autonomous Marine Operations and Systems, NTNU.*
[c] *Department of Applied Mathematics and Computer Science, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark.*

Abstract

By introducing a stage-wise prediction formulation that enables the use of highly efficient quadratic programming (QP) solution methods, this paper expands the computational toolbox for solving step response MPC problems. We propose a novel MPC scheme that is able to incorporate step response data in a traditional manner and use the computationally efficient block factorization facilities in QP solution methods. In order to solve the MPC problem efficiently, both tailored Riccati recursion and condensing algorithms are proposed and embedded into an interior-point method. The proposed algorithms were implemented in the HPMPC framework, and the performance is evaluated through simulation studies. The results confirm that a computationally fast controller is achieved, compared to the traditional step response MPC scheme that relies on an explicit prediction formulation. Moreover, the tailored condensing algorithm exhibits superior performance and produces solution times comparable to that achieved when using a condensing scheme for an equivalent (but much smaller) state-space model derived from first-principles. Implementation aspects necessary for high performance on embedded platforms are discussed, and results using a programmable logic controller are presented.

Keywords: Model Predictive Control; Step response models; Block factorization; Interior-point methods; Riccati recursion; Condensing; Numerical optimization

1. Introduction

Model Predictive Control (MPC) is an advanced control method based on numerical optimization. MPC uses a model of the plant to predict future state (or output) trajectories in a well defined constrained multivariable optimal control framework. The MPC optimization problem can be formulated as a *multistage* problem. When the plant model is linear, and a discrete state-space representation is used, the characteristic structure of the multistage optimization problem becomes apparent. The plant model equations (which become equality constraints in the optimization problem) are such that each stage equation involves coupling variables that link one stage to the next.

The capability of exploiting the multistage structure through the use of dynamic programming or block factorization techniques (e.g. Riccati recursion) was identified in [1, 2] as a key factor to consider when developing efficient MPC algorithms. This observation has led to the development of several high-speed interior-point solvers among which Fast MPC [3], FORCES [4], and HPMPC [5] are noteworthy in the context of embedded MPC. Alternative solvers that do not exploit the inherent multistage problem structure in MPC are also common. In order to use such solvers, a usual preparation step involves recasting the MPC problem as a QP problem that does not necessarily preserve the multistage structure. For instance, qpOASES [6], as well as most active-set solvers, prefer a condensed QP problem formulation where the state variables are eliminated.

For MPC problems that use step response models, the existing MPC algorithms mainly resort to compact formulations of the prediction model where one stage equation can depend on variables from all stages (see e.g. [7, 8, 9, 10]). Consequently, the choice of a QP solver for MPC schemes that use step response models presently excludes efficient solvers whose strength is their ability to exploit the multistage structure (readily apparent in MPC schemes that use state-space models).

It is possible to obtain an equivalent state-space realization from step response models [11, 12], and it is therefore possible to redesign a given step response MPC scheme to use a state-space realization instead [8]. However, in practical examples, where real (possibly noisy) plant data is involved, even very efficient and numerically stable realization algorithms resort to heuristic criteria when identifying significant states [11, 8]. As a result, the (minimal) state-space realization may have a relatively large state dimension, where the system matrices do not exhibit any obvious structure that can be exploited in a systematic way [11]. If some model reduction technique is used, validation procedures will be required to ensure that an acceptable response is produced by the resulting state-space model.

It is clear that opting for a state-space realization from step response models may not always result in easy and straightforward control design, commissioning, or maintenance proce-

Email addresses: kwame.kufoalor@itk.ntnu.no (D. K. M. Kufoalor), giaf@dtu.dk (G. Frison), lars.imsland@itk.ntnu.no (L. Imsland), tor.arne.johansen@itk.ntnu.no (T. A. Johansen), jbjo@dtu.dk (J. B. Jørgensen)

dures for industrial MPC installations. Recall that the main reason why step response models are widely accepted in industrial practice, and are still common in industrial MPC schemes, is that the step response model approach facilitates easy and intuitive identification, control design, and maintenance procedures [13, 8, 7].

The main motivation leading to the contributions in this paper is the need to fill the gap between fast QP solver developments and industrial MPC implementations based on step response models. Therefore, this paper proposes a new, but mathematically equivalent, formulation for step response MPC. The formulation facilitates the use of block factorization in the QP solution method, and it incorporates the original step response data in a traditional way. A dedicated state-space realization algorithm is not needed in the proposed MPC scheme. This implies that no extra model validation procedures are required. The implications for both Riccati recursion and condensing based solvers are studied. Discussions on implementation aspects in the HPMPC [5] framework, targeting embedded MPC applications, are also included.

Further motivation and background for the methods proposed in this paper are given in sections 3–4, and the main contributions of this work are presented in sections 5–8. Simulation results from a simple MPC problem and a more complex industrial example are discussed in sections 9–10, and concluding remarks are given in Section 11.

2. Notation and definitions

In this paper, the following notation and definitions are used.

- X represents the state vector of a state-space representation of step response models, where the vector dimension is usually larger than that of the state vector x of a corresponding state-space model derived from first-principles.
- $X(j)$ or $x(j)$ represents a state (or stage) vector for the *stage* j in a multistage problem.
- $x_i(j)$ is element i in the stage vector $x(j)$, i.e. $x(j) = \{x_i(j)\}_{i=1}^{i=n_x}$, where n_x is the number of elements in $x(j)$.
- $y(k+j|k)$ represents the prediction of $y(k+j)$ using available information at time k.
- $\bar{\cdot}$ implies that the variable, vector, or matrix belongs to the augmented state-space system, which includes the previous input as a state variable.
- $\tilde{\cdot}$ indicates that the vector or matrix belongs to the recursive state-space representation of step response models.
- $\hat{\cdot}$ implies that the value of the vector is an estimate or a prediction. For computed input moves, $\hat{u}_j := \Delta u_j$.
- $\cdot^d$ indicates that the variable or element is a *dummy* i.e. it does not change the outcome (or value) of the computation it is involved in.
- *step*-MPC (or *step*-response MPC) refers to the traditional step response based MPC scheme, where output predictions are typically computed explicitly.
- *ress*-MPC (or *realized state-space* MPC) is a state-space MPC scheme, where the state-space model is obtained from step response data, using a realization algorithm.
- *srss*-MPC (or *step-response state-space* MPC) is the new MPC scheme proposed herein, based on the recursive computation of output predictions using step response data (in a specially structured state-space representation).
- `chol(·)` represents a function that returns the Cholesky factor of the input matrix.
- *flops* is an acronym for *floating-point operations*.

3. Multistage problems and block factorization

3.1. MPC problem formulation

Industrial MPC problems are typically formulated in terms of controlled variables (CVs), disturbance variables (DVs), and manipulated variables (MVs) (see e.g. [13, 8, 7]). The CVs are usually plant outputs $y(k)$ that can be measured or estimated, DVs are measured (or estimated) disturbances $d(k)$, and the MVs are the control inputs $u(k)$. Based on these variables, an MPC problem whose objective is to track a given output reference $r_y(k)$ can be formulated as

$$\min \sum_{j=H_w}^{H_p} \|y(k+j|k) - r_y(k+j)\|_{\bar{Q}_y}^2 + \sum_{j=0}^{H_u-1} \|\Delta u(k+j)\|_{\bar{P}}^2 \quad (1a)$$

subject to

$$\underline{\Delta u} \le \Delta u(k+j) \le \overline{\Delta u}, \quad \underline{u} \le u(k+j) \le \bar{u}, \quad (1b)$$
$$\underline{y} \le y(k+j|k) \le \bar{y}, \quad (1c)$$
$$u(k+j) = u(k+j-1) + \Delta u(k+j), \quad (1d)$$
$$y(k+j|k) = \hat{y}(k+j|k), \quad (1e)$$

where $j \in \{H_w, \ldots, H_p\}$ for the output constraints, $j \in \{0, \ldots, H_u - 1\}$ for the input constraints, $H_w \ge 1$ and $H_u \le H_p$. The j-step ahead prediction of the CVs, at time k, based on the plant dynamics is represented by $\hat{y}(k+j|k)$, and the implementation of Eq. (1e) is crucial for the structure of problem (1). Furthermore, the way the predictions $\hat{y}(k+j|k)$ are made has a great effect on the performance of the closed-loop system, and the choice of prediction strategy is therefore an important point to consider when formulating the MPC problem [8].

Note that a straightforward extension of problem (1) to include soft constraints and stability terms (or stability constraints) can be made without losing the inherent multistage structure of the MPC problem. Moreover, nominal closed-loop stability can be achieved by an adequate choice of the weights $\bar{Q}_y$, $\bar{P}$, and the horizon lengths H_p and H_u (see e.g. [14]).

3.2. Effect of prediction strategy on QP problem structure

Consider the linear time-invariant (LTI) state-space model

$$x(k+1) = Ax(k) + Bu(k) + B_d d(k), \tag{2a}$$
$$y(k) = Cx(k) + Du(k) + w(k), \tag{2b}$$

where $x(k)$ is the state vector, $d(k)$ is a *known* disturbance variable, $w(k)$ is an *unknown* disturbance, and $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $B_d \in \mathbb{R}^{n_x \times n_d}$, $C \in \mathbb{R}^{n_y \times n_x}$, $D \in \mathbb{R}^{n_y \times n_u}$.

The predictions $\hat{y}(k+j|k)$, for $j = 1, \ldots, H_p$, can be computed explicitly by iterating Eq. (2). The explicit predictions provide the possibility of eliminating the states from the decision variables of Eq. (1), resulting in a dense QP problem. Although explicit predictions are used, a sparse QP formulation that keeps the states as decision variables may be preferable for some QP solver implementations. However, it can be seen in the following derivation that the multistage structure of problem (1) is lost when the explicit prediction approach is used. Specifically, each stage equation depends on one or several stage variables of $\Delta u(k+j)$.

Assume the plant dynamics can be represented by Eq. (2), with $D = 0$, and the disturbance-free case (i.e. $d(k) = 0$, $w(k) = 0$). The model can be written as

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ y(k) &= Cx(k). \end{aligned} \tag{3}$$

Simply iterating Eq. (3) leads to the explicit predictions [8]:

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \vdots \\ \hat{y}(k+H_p|k) \end{bmatrix} = \begin{bmatrix} CA & CB \\ CA^2 & C(AB+B) \\ \vdots & \vdots \\ CA^{H_p} & \sum_{i=0}^{H_p-1} CA^i B \end{bmatrix} \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}$$
$$+ \begin{bmatrix} CB & & \\ C(AB+B) & CB & \\ \vdots & \ddots & \ddots \\ \sum_{i=0}^{H_p-1} CA^i B & \cdots & \sum_{i=0}^{H_p-H_u} CA^i B \end{bmatrix} \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \vdots \\ \Delta u(k+H_u-1) \end{bmatrix}, \tag{4}$$

which is conveniently grouped into two terms. The first depends on the current augmented state $\bar{x}(k) = [x^T(k),\ u(k-1)]^T$, and the second depends on the vector of (unknown) future control actions $\Delta u(k+j)$, which are decision variables calculated in problem (1).

When step response models are used to describe the plant dynamics in MPC, the prediction model formulations found in the existing MPC literature resort to the explicit prediction approach. Specifically,

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \vdots \\ \hat{y}(k+H_p|k) \end{bmatrix} = \begin{bmatrix} \hat{y}_f(k+1|k) \\ \hat{y}_f(k+2|k) \\ \vdots \\ \hat{y}_f(k+H_p|k) \end{bmatrix}$$
$$+ \begin{bmatrix} S(1) & 0 & \cdots \\ S(2) & S(1) & \ddots \\ \vdots & \vdots & \ddots \\ S(H_p) & \ldots & S(H_p - H_u + 1) \end{bmatrix} \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \vdots \\ \Delta u(k+H_u-1) \end{bmatrix}, \tag{5}$$

where $\hat{y}_f(k+j|k)$ is the *free response* known at time k, and $S(\cdot)$ represents the step response coefficient at the corresponding sampling instant. A survey of step response prediction models can be found in [15], where it is shown that different formulations found in the MPC literature differ only in the way $\hat{y}_f(k+j|k)$ is computed. Note the structural similarity between the explicit predictions (4) and (5), which indicates that the first term of Eq. (4) is simply the free response $\hat{y}_f(k+j|k)$ in Eq. (5). The relation $S(j) = \sum_{i=0}^{j-1} CA^i B$ can also be deduced by direct comparison of Eq. (4) and Eq. (5).

From the above discussions, it is clear that the use of explicit prediction for step response models cannot take advantage of QP solver methods that are designed to exploit the multistage structure inherent in problem (1). It is known that the multistage structure of problem (1) is preserved if $\hat{y}(k+j|k)$ in Eq. (1e) is defined as [2]

$$\hat{y}(k+j|k) = \bar{C}\bar{A}\bar{x}(k+j-1|k) + \bar{C}\bar{B}\Delta u(k+j-1), \tag{6}$$

$$\text{where} \quad \bar{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} B \\ I \end{bmatrix}, \quad \bar{C} = \begin{bmatrix} C & 0 \end{bmatrix}.$$

Section 4 therefore examines different strategies that enable the use of step response data in a formulation similar to Eq. (6), and the new multistage prediction approach proposed in this paper is presented in Section 5.

3.3. Computational efficiency of multistage QP problems

The MPC problem (1) is a constrained convex QP problem, which can be solved in different ways. Using the explicit predictions, (4) or (5), either a dense or sparse QP solver that is capable of exploiting the general structure of the QP problem may be a suitable choice. The cost of solving a dense QP, where all the states are eliminated is typically $O(N_p^3 n_u^3)$. This limits the dense approach to problems with relatively small horizon, N_p, and number of inputs, n_u. When efficient sparse solver techniques are implemented, the sparse QP based on explicit predictions can be solved with $O(N_p(n_x+n_u)^3)$ complexity [1].

The possibility of achieving linear complexity in N_p for the sparse QP problem has motivated the development of different block factorization techniques for QP solution methods based on the multistage structure of problem (1), using Eq. (6). Efficient block factorization strategies are employed in both Riccati recursion based solvers (e.g. HPMPC) and Schur complement based solvers (e.g. Fast MPC and FORCES). Note that although different sparse solvers exhibit similar asymptotic complexities, their performances in practice may differ by up to an order of magnitude [16].

4. State-space realizations from step response models

The discussions in the preceding sections motivate the use of a state-space model in MPC. If step response data of the plant is available, well established realization techniques can be used to translate such data into a minimal state-space realization

[11, 12, 8]. Among the numerous existing state-space realization algorithms (see e.g. [11] for an overview), the algorithms of [17, 18, 19, 20] have proved to be practically useful and numerically reliable [11, 12, 8]. However, due to measurement noise in step response data (in practice), the exact response of the underlying LTI system can in general not be produced by a state-space realization of the same order [11]. This implies that realization algorithms may rely on some heuristic criteria in order to arrive at good low-order approximations.

The resulting system matrices A, B, C, D, are in general dense and do not exhibit any specific structure [11, 8], even though canonical minimal realizations are possible in some cases [21]. Furthermore, the state-space models obtained from realization algorithms do not have any direct physical interpretation, and it will be shown (later in this paper) that an observer is required in order to relate the states generated by realization algorithms to measured values of controlled variables in existing industrial MPC schemes. In order to facilitate the incorporation of state-space realization algorithms into an automatic code-generation framework, it is desirable to use realization strategies that do not rely on heuristics, and therefore avoid extra manual verification or validation procedures. This implies that relatively large state dimensions are expected, typically $N \cdot \min(n_u, n_y)$ [8].

An alternative approach to the use of dedicated state-space realization algorithms, is to use the direct state-space interpretation of step response models (see e.g. [10, 9], or [7]). In this way, no further approximation is introduced to the step response data, and the state variables can be interpreted as future outputs [10, 9], or as a vector of past outputs and control moves [7]. Both interpretations lead to state-space models that have relatively large state vector dimensions. However, the interpretation of [10] comes with the advantage that the system matrices $\tilde{A}$ and $\tilde{C}$ are sparse and have fixed structure that can be carefully exploited in an MPC algorithm:

$$\tilde{X}(k+1) = \tilde{A}\tilde{X}(k) + \tilde{B}\Delta u(k), \tag{7a}$$

$$\tilde{y}(k) = \tilde{C}\tilde{X}(k), \tag{7b}$$

where $\tilde{C} = \begin{bmatrix} I_{n_y} & 0 & \dots & 0 \end{bmatrix}_{n_y \times N}$,

$$\tilde{A} = \begin{bmatrix} 0 & I_{n_y} & 0 & \cdots & 0 \\ 0 & 0 & I_{n_y} & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I_{n_y} \\ 0 & 0 & 0 & \cdots & I_{n_y} \end{bmatrix}_{n_y N \times n_y N} \qquad \tilde{B} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{N-1} \\ S_N \end{bmatrix}_{n_y N \times n_u}$$

The state vector, $\tilde{X}(k)$ of dimension $N \cdot n_y$, is defined as

$$\tilde{X}(k) \triangleq \tilde{Y}(k) = \begin{bmatrix} \tilde{y}^T(k) & \tilde{y}^T(k+1) & \dots & \tilde{y}^T(k+N-1) \end{bmatrix}^T \tag{8}$$

and the matrix S_i, for $i = 1, \dots, N$, contains the step response coefficients relating each input-output pair. Model (7) uses the assumption $S_{N+1} \approx S_N$, which is valid for stable plants. In order to represent integrating plants, the extensions in [9] can be used.

5. Multistage QP using step response models

A multistage problem based on the recursive computations in problem (7) and structural exploitation techniques for step response MPC are discussed in this section. The formulation used and the resulting algorithms provide a new MPC scheme, termed *srss*-MPC. In order to highlight the advantages of the *srss*-MPC scheme, comparisons are made with the *ress*-MPC scheme, which is based on the use of realization algorithms. Since both MPC schemes involve the use of predictions in the form of Eq. (6), the derivations in this and the following sections are compared to the general multistage QP formulation derived from the MPC problem (1) using Eq. (6). The general multistage QP problem is provided in Appendix A.

5.1. Multistage step response prediction

The main results of this work are based on the observation that a multistage prediction strategy can be derived for step response MPC when the free response trajectory is used to initialize the recursive computations of the future output predictions. Using the free response vector $\hat{y}_f(k+j|k)$ as the initial state, the stage-wise predictions of the *srss*-MPC can be derived from the traditional explicit predictions (5) in a straightforward manner.

Consider $j = 0, \dots, N_p$, which includes the initial stage $j = 0$, defined in the multistage problem (A.1). Let $N_p = 3$, without loss of generality. From Eq. (5),

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \hat{y}(k+3|k) \end{bmatrix} = \begin{bmatrix} \hat{y}_f(k+1|k) \\ \hat{y}_f(k+2|k) \\ \hat{y}_f(k+3|k) \end{bmatrix} + \begin{bmatrix} S_1 & & \\ S_2 & S_1 & \\ S_3 & S_2 & S_1 \end{bmatrix} \begin{bmatrix} \Delta u(k+0) \\ \Delta u(k+1) \\ \Delta u(k+2) \end{bmatrix}$$

which is the sum of the free response $\hat{y}_f(k+j|k)$ and the *forced response* (that depends on the current and future control moves $\Delta u(k+j)$). In order to compute the forced response (and therefore the predictions) recursively, consider rewriting the explicit predictions as

$$\begin{bmatrix} \hat{y}(k+1|k) \\ \hat{y}(k+2|k) \\ \hat{y}(k+3|k) \end{bmatrix} = \begin{bmatrix} \hat{y}_f(k+1|k) \\ \hat{y}_f(k+2|k) \\ \hat{y}_f(k+3|k) \end{bmatrix} + \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} \Delta u(k+0) + \begin{bmatrix} 0 \\ S_1 \\ S_2 \end{bmatrix} \Delta u(k+1) + \begin{bmatrix} 0 \\ 0 \\ S_1 \end{bmatrix} \Delta u(k+2).$$

It is now obvious that, in order to compute all terms depending on $\Delta u(k+j)$ at each stage j, a state vector $\tilde{X}(k+j)$ can be used to accumulate the partial sums, as

$$\begin{aligned} \begin{bmatrix} \tilde{x}_1(k+1) \\ \tilde{x}_2(k+1) \\ \tilde{x}_3(k+1) \end{bmatrix} &= \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \tilde{x}_1(k+0) \\ \tilde{x}_2(k+0) \\ \tilde{x}_3(k+0) \end{bmatrix} + \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} \Delta u(k+0) \\ \begin{bmatrix} \tilde{x}_1(k+2) \\ \tilde{x}_2(k+2) \\ \tilde{x}_3(k+2) \end{bmatrix} &= \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \tilde{x}_1(k+1) \\ \tilde{x}_2(k+1) \\ \tilde{x}_3(k+1) \end{bmatrix} + \begin{bmatrix} 0 \\ S_1 \\ S_2 \end{bmatrix} \Delta u(k+1) \\ \begin{bmatrix} \tilde{x}_1(k+3) \\ \tilde{x}_2(k+3) \\ \tilde{x}_3(k+3) \end{bmatrix} &= \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \tilde{x}_1(k+2) \\ \tilde{x}_2(k+2) \\ \tilde{x}_3(k+2) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ S_1 \end{bmatrix} \Delta u(k+2) \end{aligned} \tag{9}$$

where $\tilde{X}(k+j) = \begin{bmatrix}\tilde{x}_1^T(k+j) & \tilde{x}_2^T(k+j) & \tilde{x}_3^T(k+j)\end{bmatrix}^T$, $A(j)$ is the identity matrix, $B(j)$ is time-variant and contains the step response coefficients, and the initial state is the free response

$$\hat{X}(0) := \tilde{X}(k+0) = \begin{bmatrix}\tilde{x}_1(k+0)\\ \tilde{x}_2(k+0)\\ \tilde{x}_3(k+0)\end{bmatrix} = \begin{bmatrix}\hat{y}_f(k+1|k)\\ \hat{y}_f(k+2|k)\\ \hat{y}_f(k+3|k)\end{bmatrix}.$$

The output predictions are then trivially computed as

$$\begin{aligned}\hat{y}(k+1|k) &= \begin{bmatrix}I & 0 & 0\end{bmatrix}\tilde{X}(k+1)\\ \hat{y}(k+2|k) &= \begin{bmatrix}0 & I & 0\end{bmatrix}\tilde{X}(k+2)\\ \hat{y}(k+3|k) &= \begin{bmatrix}0 & 0 & I\end{bmatrix}\tilde{X}(k+3)\end{aligned} \tag{10}$$

and the derived stage-wise prediction equations, (9) and (10), are in the state-space form

$$\begin{aligned}\tilde{X}(k+j+1) &= \tilde{A}(j)\tilde{X}(k+j) + \tilde{B}(j)\Delta u(k+j)\\ \hat{y}(k+j|k) &= \tilde{C}(j)\tilde{X}(k+j).\end{aligned}$$

Note that the component of $\tilde{X}(k+j)$ with index 1 is not updated after the first stage (i.e. $\tilde{x}_1(k+1) = \tilde{x}_1(k+2) = \tilde{x}_1(k+3)$), and the component with index 2 is not updated after the second stage (i.e. $\tilde{x}_2(k+2) = \tilde{x}_2(k+3)$). This observation (and other related properties summarized in Section 5.2) form the basis of the structural exploitation techniques proposed in this paper.

It may be useful to explicitly include the expression of the future control values $u(k+j)$ as optimization variables. This can be easily obtained by augmenting the state as

$$\begin{bmatrix}\tilde{x}_1(k+1)\\ \tilde{x}_2(k+1)\\ \tilde{x}_3(k+1)\\ u(k+0)\end{bmatrix} = \begin{bmatrix}I & 0 & 0 & 0\\ 0 & I & 0 & 0\\ 0 & 0 & I & 0\\ 0 & 0 & 0 & I\end{bmatrix}\begin{bmatrix}\tilde{x}_1(k+0)\\ \tilde{x}_2(k+0)\\ \tilde{x}_3(k+0)\\ u(k-1)\end{bmatrix} + \begin{bmatrix}S_1\\ S_2\\ S_3\\ I\end{bmatrix}\Delta u(k+0)$$

illustrated for $j=0$. The state vector size becomes $n_X = N_p n_y + n_u$. Note that the diagonal structure of the $A(j)$ matrices is preserved.

To sum up, general equations for the multistage step response predictions in Eq. (9) and (10) are

$$\tilde{x}_i(k+j+1) = \tilde{x}_i(k+j) + S_{i-j}\Delta u(k+j), \quad j = 0,\dots,N_p \tag{11}$$

$$\hat{y}(k+j|k) = \tilde{x}_j(k+j), \quad j = 1,\dots,N_p \tag{12}$$

where $S_{i-j} = 0$ for $i \le j$, and $i = 1,\dots,N_p$.

5.2. *Exploiting the system dynamics structure*

In this subsection, the inherent structure of the recursive computations presented in Section 5.1 are discussed, and efficient implementation strategies are proposed based on the observation that only some components of $\tilde{X}(k+j)$ are updated from one stage to the next. In order to facilitate straightforward referencing, the inherent structural properties in Eq. (9) or (11) are summarized in the following:

OB1 The first variable $\tilde{x}_1(k+j)$ in $\tilde{X}(k+j)$ at any stage j is not required to compute the state variables at the next stage.

OB2 The number of states n_X that contribute in calculating the state vector $\tilde{X}(k+j)$ at each stage j decreases by n_y from one stage to the next, i.e. $n_X(k+0) = N_p n_y$, $n_X(k+1) = (N_p-1)n_y$, $\dots, n_X(k+N_p) = n_y$

OB3 The number of step response matrices S_i required at each stage decreases by 1 from one stage to the next, i.e. the sequence $\{S_i\}_{i=1}^{N_p-j}$, $j = 0,\dots N_p-1$, is required.

OB4 The output estimate at each stage is given by $\hat{y}(k+j|k) = \tilde{x}_j(k+j)$, $j = 1,\dots N_p$.

Based on OB1–OB4, implementation strategies that aim at attaining a high structural exploitation level in the QP solution method for the *srss*-MPC scheme can be derived. Using OB1, and considering $k=0$, the second and third equations in (9) can be rewritten as

$$\begin{bmatrix}\tilde{x}_1^d(2)\\ \tilde{x}_2(2)\\ \tilde{x}_3(2)\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & I & 0\\ 0 & 0 & I\end{bmatrix}\begin{bmatrix}\tilde{x}_1(1)\\ \tilde{x}_2(1)\\ \tilde{x}_3(1)\end{bmatrix} + \begin{bmatrix}0\\ S_1\\ S_2\end{bmatrix}\Delta u(1)$$

$$\begin{bmatrix}\tilde{x}_1^d(3)\\ \tilde{x}_2^d(3)\\ \tilde{x}_3(3)\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & I\end{bmatrix}\begin{bmatrix}\tilde{x}_1^d(2)\\ \tilde{x}_2(2)\\ \tilde{x}_3(2)\end{bmatrix} + \begin{bmatrix}0\\ 0\\ S_1\end{bmatrix}\Delta u(2)$$

$$\implies \tilde{X}_{j+1} = \tilde{A}_j\tilde{X}_j + \tilde{B}_j\hat{u}_j, \text{ cf. Eq. (A.1c)},$$

where the superscript ^{d} implies the element contains a *dummy* value, and can therefore be ignored. Therefore, $\tilde{A}_j = \tilde{A}$ can be considered either as a *diagonal* matrix or the *identity* matrix. Both options lead to a system structure that can be easily exploited in any solver that makes computations directly involving $\tilde{A}_j$.

Using $\tilde{A}_j = I$ implies $\tilde{X}_{j+1} = I\tilde{X}_j + \tilde{B}_j\hat{u}_j$, where the state vector $\tilde{X}_j$ has a fixed length $n_{\tilde{X}} = n_y N_p$. However, an easy way to keep track of the useful information in the recursive computations (i.e. identifying dummy elements) is to maintain the zeros in the diagonal of $\tilde{A}_j$. In this way more structure is introduced, which may facilitate further exploitation in some QP solution methods (including the methods used in this paper). A general multistage step response equation that is equivalent to maintaining the zeros in the diagonal of the matrix formulation above is

$$\tilde{x}_i(k+j+1) = \begin{cases}0 & \text{if } i \le j,\\ \tilde{x}_i(k+j) + S_{i-j}\Delta u(k+j) & \text{if } i > j.\end{cases}$$

Apart from the possibility of storing $\tilde{A}$ as an identity matrix, or directly incorporating $\tilde{A} = I$ into the solver algorithm where applicable, a large memory workspace may be required in a naive implementation (considering n_X states at N_p stages). However, since the applicability of multistage problems is not limited to stage variables with fixed dimensions, the dummy elements can be omitted in the prediction equations at each stage. The state dimension will then decrease from one stage to the next, as $n_X(j) = n_y(N_p - j)$, which is in agreement with OB2. In this way only the required data and state information for each stage are stored, significantly reducing the memory requirements.

In some implementation frameworks (e.g. HPMPC), it maybe preferable to reverse the state vector $\tilde{X}_j$. This will move all

useful data of the system matrices to the top rows, making it easier to keep track of useful data blocks, i.e.

$$\begin{bmatrix}\tilde{x}_3(2)\\ \tilde{x}_2(2)\end{bmatrix} = \left[\begin{array}{cc|c} I & 0 & 0\\ 0 & I & 0\end{array}\right] \left[\begin{array}{c}\tilde{x}_3(1)\\ \tilde{x}_2(1)\\ \hline \tilde{x}_1(1)\end{array}\right] + \begin{bmatrix}S_2\\ S_1\end{bmatrix}\Delta u(1)$$

$$\begin{bmatrix}\tilde{x}_3(3)\end{bmatrix} = \left[\begin{array}{cc|c} I & 0 & 0\end{array}\right] \left[\begin{array}{c}\tilde{x}_3(2)\\ \tilde{x}_2(2)\end{array}\right] + \begin{bmatrix}S_1\end{bmatrix}\Delta u(2)$$

The varying stage vector size is $n_X(j) = n_y(N_p - j)$, and the useful stage data size is $n_X^{da}(j) = n_y(N_p - j - 1)$.

Note that using a varying stage vector size implies that the general equation (11) for the multistage step response predictions is implemented only for the case where $i > j$.

5.3. Implications of initializing with the free response

Several implications follow from the particular choice of $y_f(k+j)$ as $\hat{X}(0)$. First of all, any of the existing (i.e. traditional) approaches for computing the free response can be used to obtain $\hat{X}(0)$ (see e.g. [15, 7, 8]). Moreover, the results of [15] show that an implementation using the recursive formulation (7), yields the most efficient free response computations. In this work, the free response is computed as follows:

Based on Eq. (7), $\tilde{X}(k) = \tilde{A}\tilde{X}(k-1) + \tilde{B}\Delta u(k-1)$, can be easily computed in an MPC scheme since estimates of both

$$\tilde{X}(k-1) = \begin{bmatrix}\tilde{y}^T(k-1|k-1) & \dots & \tilde{y}^T(k+N_p-1|k-1)\end{bmatrix}^T$$

and $\Delta u(k-1)$ are available at time k. Simply shifting $\tilde{X}(k)$ one step ahead (i.e. assuming $\Delta u(k) = 0$) produces the free response $\hat{X}_f(k+1) = \{\hat{y}_f(k+j|k),\ j = 1,\dots,N_p\}$, which is usually corrected by some disturbance estimate $V(k)$, i.e.

$$\hat{X}_f(k+1) = \tilde{A}_p\tilde{X}(k) + V(k), \tag{13}$$

where $\tilde{A}_p$ contains the first N_p rows of $\tilde{A}$ in Eq. (7).

Another implication is that since the disturbance model $V(k)$ is included in the free-response computations, the initial state $\tilde{X}(0)$ describes the response, including the predicted effect of disturbances, when no future control actions are applied to the plant. This implies that the estimate of disturbances are computed only once in each sampling interval, outside the QP solver.

Compared to the option of using the realized state-space scheme (*ress*-MPC), the following remarks can be made. In the *ress*-MPC, further augmentation of the system dynamics model may be derived in order to incorporate an appropriate disturbance model. Known disturbances $d(k)$ can be implemented as extra (possibly time-varying) parameters in the system dynamics equation. For the case where a constant disturbance model is used for $w(k)$ in Eq. (2) (i.e. the typical choice in industrial MPC schemes [8]), the cost term involving the output $y(j)$ in problem (1) can be redefined as

$$\begin{aligned} y^T(j)\bar{Q}_y y(j) &= (Cx(j) + w(0))^T\bar{Q}_y(Cx(j) + w(0)) \\ &= x^T(j)C^T\bar{Q}_yCx(j) + 2w^T(0)\bar{Q}_yCx(j), \end{aligned}$$

and the output is calculated as $y(j) = Cx(j) + w(0)$. This is possible since $w(j) = w(0)$ over the stages $j = 1,\dots,N_p$ in the multistage problem, and can therefore be interpreted as a constant parameter in the QP solver.

Clearly, a state observer is required for the *ress*-MPC scheme, since outputs $y(k)$, instead of states, are measured, and C is generally not equal to I. Even if $C = I$, the presence of output disturbances $w(k)$ makes an observer necessary.

5.4. Control objective and constraints

The relations that convert the original MPC problem (1) to the multistage QP problem (A.1), stated in Appendix A, apply for both the *ress*-MPC and *srss*-MPC schemes. Moreover, a straightforward design parameter translation from a traditional step response MPC scheme is achieved when the *srss*-MPC is used. Since the state vector $\tilde{X}(k)$ consists of outputs, an extra set-point filter that ensures the feasibility of the output reference trajectory $r_y(k+j)$ in terms of state and input variables (i.e. $r_X(k+j)$ and $r_u(k+j)$) is not needed for the *srss*-MPC scheme. Specifically, $r(j) = r_y(k+j)$ in problem (A.1) for *srss*-MPC, whereas $r(j) = [r_x^T(k+j),\ r_u^T(k+j)]^T$ for the *ress*-MPC.

Therefore, it is straightforward to derive the matrices of the cost function of the *srss*-MPC according to problem (A.1):

$$\tilde{Q}(1) = 2\begin{bmatrix}\bar{Q}_y & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & \bar{R}\end{bmatrix},\quad \tilde{q}(1) = -\begin{bmatrix}\bar{Q}_y r_y(k+1)\\ 0\\ 0\\ \bar{r}\end{bmatrix},$$

$$\tilde{Q}(2) = 2\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & \bar{Q}_y & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & \bar{R}\end{bmatrix},\quad \tilde{q}(2) = -\begin{bmatrix}0\\ \bar{Q}_y r_y(k+2)\\ 0\\ \bar{r}\end{bmatrix},$$

$$\tilde{Q}(3) = 2\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & \bar{Q}_y & 0\\ 0 & 0 & 0 & \bar{R}\end{bmatrix},\quad \tilde{q}(3) = -\begin{bmatrix}0\\ 0\\ \bar{Q}_y r_y(k+3)\\ \bar{r}\end{bmatrix},$$

$$\tilde{R}(j) = 2\bar{P},\quad \tilde{r}(j) = r_y(k+j),\quad j = 0,1,2.$$

where box constraints on input values are translated into the penalties $\bar{R}$ and $\bar{r}$, which provide a convenient formulation in an interior-point method (IPM) framework. Moreover, the sparsity pattern observed in the cost function matrices can be exploited in some QP solution methods (see Section 6.2 for a straightforward approach). Note that the above matrices may be defined according to the size-varying $\tilde{X}(j)$. Moreover, computations that do not involve the system dynamics in the QP solver can avoid using size-varying variables, since the computed n_y output predictions $\hat{y}(k+j|k)$ at each stage j can be used directly.

Furthermore, since $C(j)$ extracts the current output, computed for each stage using identity matrices, simple box constraints on the output remain unchanged in the *srss*-MPC scheme. The particular sparse structure obtained when inequality (A.1d) maintains the original box constraints from (1c) can be easily exploited in the QP solver method. For instance, the box constraints can be easily translated into penalties in the cost function in order to simplify the solution method. For the *ress*-MPC,

on the other hand, the elements of $C(j)$ can hold any real number, implying that simple box constraints on outputs translate into general constraints on states.

5.5. Problem size reduction strategies

Since long prediction horizons are usually chosen in practice, and the step response models tend to be large, the use of techniques such as MV blocking and CV evaluation points are typical in industrial MPC schemes. Due to the stage-wise prediction of formulation (A.1), a straightforward translation of an MPC problem setup that implements MV blocking is to enforce $\Delta u_j = 0$ for the indexes j where MVs are no allowed to change. In contrast to the use of explicit predictions, the stage-wise approach does not directly eliminate the corresponding Δu variables from the QP problem. Enforcing MV blocks in the *srss*-MPC and *ress*-MPC schemes introduces the possibility of exploiting the blocking information in the solution method, since the system dynamics equation (A.1c) reduces to $\bar{x}_{j+1} = \bar{A}_j \bar{x}_j$ at the stages where control input moves are blocked.

Moreover, a strategy that implements CV evaluation points can be easily derived for the *srss*-MPC by exploiting the fact that A_j simply shifts the state vector $\tilde{X}_j$ to $\tilde{X}_{j+1}$ if $\Delta u_j = 0$. Note that $\hat{u}_j := \Delta u_j = 0$ at points where CVs are not evaluated. Consider evaluation points $\{\ell = 1,\ 2\}$ corresponding to $\{j = 1,\ 4\}$ in the horizon, and allow inputs to change at $\{j = 0,\ 1\}$. It is then easy to verify that

$$
\begin{array}{lll}
\tilde{X}_1{=}\tilde{A}_0\tilde{X}_0+\tilde{B}_0\hat{u}_0 \ \rightarrow & \tilde{X}_1{=}\tilde{A}_0\tilde{X}_0+\tilde{B}_0\hat{u}_0 \ \rightarrow & \tilde{X}_1{=}\tilde{A}_0\tilde{X}_0+\tilde{B}_0\hat{u}_0 \\
\tilde{X}_2{=}\tilde{A}_1\tilde{X}_1+\tilde{B}_1\hat{u}_1 & \tilde{X}_2{=}\tilde{A}_1\tilde{X}_1+\tilde{B}_1\hat{u}_1 & \tilde{X}_4{=}\tilde{A}_3(\tilde{X}_1+\tilde{B}_1\hat{u}_1) \\
\tilde{X}_3{=}\tilde{A}_2\tilde{X}_2+\tilde{B}_2\hat{u}_2 & \tilde{X}_3{=}\tilde{A}_2(\tilde{X}_1+\tilde{B}_1\hat{u}_1) & \\
\tilde{X}_4{=}\tilde{A}_3\tilde{X}_3+\tilde{B}_3\hat{u}_3 & \tilde{X}_4{=}\tilde{A}_3(\tilde{X}_1+\tilde{B}_1\hat{u}_1) &
\end{array}
$$

$$
\implies \tilde{X}_{\xi(\ell)} = \tilde{A}_{\xi(\ell)-1}(\tilde{X}_{\xi(\ell-1)} + \tilde{B}_{\xi(\ell-1)}\hat{u}_{\xi(\ell-1)}),
$$

where $\xi(\ell)$ is a function that returns the corresponding stage index j of the evaluation point index ℓ. The resulting system of stage equations has the same structure as the original recursive system (9). Since the corresponding outputs at the points not evaluated in the prediction horizon can be omitted from the state vector $\tilde{X}_{\xi(\ell)}$, the state dimension reduces to $n_X = n_{Ev} n_y$, where n_{Ev} is the number of evaluation points. It is also straightforward to extract the corresponding step response data for the reduced $\tilde{B}_{\xi(\ell-1)}$ matrices. Note that, since the state vector $\tilde{X}$ of the *srss*-MPC consists of future outputs that describe the response of the plant at specific points in the prediction horizon, the above derivation implies that the same evaluation points selected for a traditional step response MPC application can be applied directly to the *srss*-MPC.

Using a similar strategy for the *ress*-MPC will require the computation of the corresponding $\bar{A}_{\xi(\ell)-1}$ at each evaluation point. For instance, the last stage in the above illustration will become

$$
\begin{array}{l}
\bar{x}_4{=}\bar{A}_3\bar{x}_3+\bar{B}_3\hat{u}_3 \ \rightarrow \ \bar{x}_4{=}\bar{A}_3\bar{A}_2(\bar{A}_1\bar{x}_1+\bar{B}_1\hat{u}_1) \ \rightarrow \\
\bar{x}_4{=}\bar{A}^3\bar{x}_1+\bar{A}^2\bar{B}\hat{u}_1 \\
\implies \tilde{X}_{\xi(\ell)} = \bar{A}^{\xi(\ell)-1}\tilde{X}_{\xi(\ell-1)} + \bar{A}^{\xi(\ell)-2}\tilde{B}_{\xi(\ell-1)}\hat{u}_{\xi(\ell-1)},
\end{array}
$$

considering minimal LTI models realized from step response data. It is obvious that numerical problems may arise when large powers of $\bar{A}$ are involved. The relation between states and outputs at the evaluation points must also be considered for the *ress*-MPC. It is therefore not a trivial task to implement the evaluation points selected for a traditional step response MPC in the *ress*-MPC scheme.

6. Impact on different solver frameworks

In this section different solver frameworks for the solution of the *srss*-MPC problem are presented.

Interior-point methods (IPM) are considered for the solution of the QP problems. Beside the usual good features of IPM solvers in an MPC framework (reliability, low and almost constant number of iterations, possibility to have high-accuracy solutions), IPM solvers are chosen also because of their ability to efficiently deal with specialized constraints such as hard or soft box constraints (that are preserved in the *srss*-MPC scheme, as shown in Section 5.4).

In an IPM algorithm, the most computationally expensive operation is the solution of the unconstrained sub-problems to compute the Newton direction. In this section two solver frameworks are considered for the solution of the unconstrained sub-problems: Riccati recursion and condensing methods.

It is important to note that condensing is used as a technique to speed up the solution of the unconstrained problem and therefore performed on-line at each IPM iteration. Therefore the outputs are retained as optimization variables in the IPM algorithms, and the box constraints on outputs are retained in the optimization problem, and not transformed into general polytopic constraints on the inputs. This choice is justified by the fact that the condensing procedure is computationally cheap compared to the Riccati recursion for the *srss*-MPC scheme.

6.1. Computational speedup in a Riccati framework

The Riccati recursion is a well known method for the solution of unconstrained MPC problems, as it can efficiently exploit their special structure [2]. For simplicity, and considering the fact that the Newton direction in an IPM can be computed by solving an instance of the linear quadratic regulator (LQR) problem, the Riccati recursion computations are summarized in Alg. 1 for the underlying LQR problem of *srss*-MPC. Alg. 1 can be embedded in an IPM using a similar procedure as that described in [2] (i.e. the Riccati recursion retains its structure in the IPM, and corresponding translations for $\tilde{Q}_j$, $\tilde{R}_j$, and $\tilde{r}_j$ are used).

The Riccati recursion has an asymptotic cost of $O(N_p(n_X^3 + n_X^2 n_u + n_X n_u^2 + n_u^3))$ flops, where n_X is the state vector size, n_u is the control vector size and N_p is the prediction/control horizon length. In particular, the $O(N_p n_X^3)$ term (i.e. the term cubic in the state vector dimension) comes from the computation of the term $\tilde{A}_j^T P_{j+1}^T \tilde{A}_j$ in line 6 of Alg. 1. Since $n_X = N_p n_y$, the cost of the Riccati recursion is $O(N_p^4 n_y^3 + N_p^3 n_y^2 n_u + N_p^2 n_y n_u^2 + N_p n_u^3)$ flops when computed as a function of n_y, n_u and N_p. This

means that, if a generic Riccati recursion solver is employed, the computational complexity grows with the 4-th power of N_p.

However, in a Riccati framework the special structure of the dynamic system equations presented in Section 5.2 can be exploited to reduce the computational complexity of the algorithm. These savings are mainly due to the following:

- Diagonal A_j matrices. This means that the term $\tilde{A}_j^T P_{j+1}^T \tilde{A}_j$ can be computed in time $O(N_p n_X^2)$, reducing the asymptotic complexity of the algorithm. Similarly, the term $(P_{j+1}^T \tilde{B}_j)^T \tilde{A}_j$ in line 5 of Alg. 1 can be computed in time $O(N_p n_X^2 n_u)$, further reducing the computational cost. The fact that the Riccati recursion becomes quadratic in n_X is especially useful since the state vector size is relatively large (i.e. $n_X = N_p n_y$) compared to that of the corresponding state-space model derived from first principles. This reduces the computational cost of the Riccati recursion to $N_p(3n_X^2 n_u + 2n_X n_u^2 + \frac{1}{3}n_u^3)$ flops, or $3N_p^3 n_y^2 n_u + 2N_p^2 n_y n_u^2 + \frac{1}{3}N_p n_u^3$ flops when computed as a function of n_y, n_u and N_p, that is cubic in N_p. Furthermore, this reduces the memory requirement to store the A_j matrices.

- Size-varying n_X. By exploiting OB2, it is possible to let the size of the state vector change, and reduce it by n_y at each stage starting from $N_p n_y$ at the first stage 0 down to n_y at the last stage N_p. The Riccati recursion can be modified to take the size-varying n_X into account, and therefore reduce both the computational cost and the memory requirements. In particular, the cost of the terms quadratic in n_X (i.e. in the form $c_1 N_p^3 n_y^2 n_u$ if $n_X = N_p n_y$, where c_1 is a positive constant) becomes

$$\sum_{n_X=n_y}^{N_p n_y} c_1 n_X^2 n_u \approx \tfrac{1}{3} c_1 N_p^3 n_y^2 n_u$$

  and therefore reduced by a factor 3. Similarly, the cost of the terms linear in n_X (i.e. in the form $c_2 N_p^2 n_y n_u^2$ if $n_X = N_p n_y$, where c_2 is a positive constant) becomes

$$\sum_{n_X=n_y}^{N_p n_y} c_2 n_X n_u^2 \approx \tfrac{1}{2} c_2 N_p^2 n_y n_u^2$$

  and therefore reduced by a factor 2. Therefore the implemented algorithm has a computational cost of $N_p^3 n_y^2 n_u + N_p^2 n_y n_u^2 + \frac{1}{3}n_u^3$ flops.

Note that there is no need to retain all states as optimization variables in the IPM: at each stage j, only the n_y states corresponding to output predictions $\hat{y}(k+j|k)$ and the n_u states corresponding to inputs $u(k+j|k)$ are needed. This simplifies the design of an efficient IPM, that does not need to deal with size-varying n_X and the sparsity structure of the state constraints. In this way, besides the cubic cost to compute the Newton direction, all other operations in the IPM algorithm can be performed in $O(N_p)$ flops.

Algorithm 1 *Tailored Riccati recursion scheme for *srss*-MPC ($\tilde{A}_j$ is diagonal, n_X is varying according to discussions the in sections 5.2 and 6.1)

Require: $\tilde{A}_j, \tilde{B}_j, \tilde{Q}_j, \tilde{R}_j, \tilde{r}_j = 0, \{j = 1,\dots,N_p\}, \hat{X}_0 := \hat{X}_f(k+1)$

1: $P_{N_p} \leftarrow \tilde{Q}_{N_p}$
2: **for** $j = N_p - 1 \to 0$ **do**
3: $\quad R_{\Lambda,j} \leftarrow \tilde{R}_j + \tilde{B}_j^T \cdot (P_{j+1}^T \cdot \tilde{B}_j)$
4: $\quad \Lambda_j \leftarrow \texttt{chol}(R_{\Lambda,j})$
5: $\quad L_j \leftarrow \Lambda_j^{-1} \cdot (P_{j+1}^T \tilde{B}_j)^T \cdot \tilde{A}_j \quad \{\text{exploit } \tilde{A}_j\}$
6: $\quad P_j \leftarrow \tilde{Q}_j + \tilde{A}_j^T \cdot P_{j+1}^T \cdot \tilde{A}_j - L_j^T L_j \quad \{\text{exploit } \tilde{A}_j\}$
7: **end for**
8: **for** $j = 0 \to N_p - 1$ **do**
9: $\quad \hat{u}_j \leftarrow -(\Lambda_j^T)^{-1} L_j \cdot \tilde{X}_j$
10: $\quad \tilde{X}_{j+1} \leftarrow \tilde{A}_j \cdot \tilde{X}_j + \tilde{B}_j \cdot \hat{u}_j \quad \{\text{exploit } \tilde{A}_j\}$
11: **end for**

* suitable for embedding into an IPM framework

6.2. Computational speedup in a condensing framework

By using the special structure of the matrices $\tilde{A}(j)$, $\tilde{B}(j)$, $\tilde{C}(j)$, $\tilde{Q}(j)$, $\tilde{R}(j)$, $\tilde{q}(j)$, $\tilde{r}(j)$, it is possible to write an efficient condensing (or state elimination) algorithm, that has the same computational complexity, i.e. $O(N_p^3(n_y n_u^2 + n_u^3) + N_p^2 n_y^2 n_u)$ as a condensing algorithm for the *original* state-space system (3). The original system refers to a state-space system derived from first principles, and it is assumed that the number of outputs n_y is equal to the number of states n_x of the original system. This assumption represents the worst case setup, implying that a faster algorithm will be achieved for a system with $n_y < n_x$.

The condensed Hessian matrix is $\tilde{H} = \tilde{\mathscr{R}} + \Gamma_u^T \tilde{\mathscr{Q}} \Gamma_u$, where

$$\tilde{\mathscr{R}} = \begin{bmatrix} \tilde{R}(0) & & \\ & \tilde{R}(1) & \\ & & \tilde{R}(2) \end{bmatrix}, \quad \tilde{\mathscr{Q}} = \begin{bmatrix} \tilde{Q}(1) & & \\ & \tilde{Q}(2) & \\ & & \tilde{Q}(3) \end{bmatrix}.$$

The matrix Γ_u is

$$\Gamma_u = \begin{bmatrix} \tilde{B}(0) & & \\ \tilde{A}(1)\tilde{B}(0) & \tilde{B}(1) & \\ \tilde{A}(2)\tilde{A}(1)\tilde{B}(0) & \tilde{A}(2)\tilde{B}(1) & \tilde{B}(2) \end{bmatrix} = \left[\begin{array}{c|c|c} S_1 & & \\ S_2 & & \\ S_3 & & \\ I & & \\ \hline 0 & 0 & \\ S_2 & S_1 & \\ S_3 & S_2 & \\ I & I & \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ S_3 & S_2 & S_1 \\ I & I & I \end{array}\right]$$

The matrix $\tilde{\mathscr{Q}}\Gamma_u$ is

$$\tilde{\mathscr{Q}}\Gamma_u = \left[\begin{array}{c|c|c} \bar{Q}_y S_1 & & \\ 0 & & \\ 0 & & \\ \bar{R} & & \\ \hline 0 & 0 & \\ \bar{Q}_y S_2 & \bar{Q}_y S_1 & \\ 0 & 0 & \\ \bar{R} & \bar{R} & \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ \bar{Q}_y S_3 & \bar{Q}_y S_2 & \bar{Q}_y S_1 \\ \bar{R} & \bar{R} & \bar{R} \end{array}\right]$$

The key concept of this condensing method is that in the $\tilde{\mathscr{Q}}\Gamma_u$ matrix, each block has a number of non-zero elements that does not depend on N_p, and therefore it is possible to compute this matrix efficiently in $O(N_p^2 n_y^2 n_u)$ flops ($O(N_p^2 n_y n_u)$ if $\bar{Q}_y$ is diagonal).

The $\tilde{H}$ matrix can be computed as $\tilde{H} = \tilde{\mathscr{R}} + \Gamma_u^T \cdot (\tilde{\mathscr{Q}}\Gamma_u)$ (and taking into account the zero sparsity pattern of the matrices) in time $O(\frac{1}{3}N_p^3 n_y n_u^2)$. This complexity could be reduced to $O(N_p^2 n_y n_u^2)$ by exploiting the fact that $\bar{Q}_y$ is the same at all stages. However, in an IPM framework this is not the case, since the cost function matrices are updated with a penalty that in general is different at each stage. Finally, the $\tilde{H}$ matrix can be factorized in time $O(\frac{1}{3}N_p^3 n_u^3)$ by means of a Cholesky factorization (see e.g. [22]).

The condensed Jacobian vector is $\tilde{g} = \tilde{r} + \Gamma_u^T \tilde{q} + \Gamma_u^T \tilde{\mathscr{Q}}\Gamma_b$, where

$$\tilde{q} = \begin{bmatrix} \tilde{q}(1) \\ \tilde{q}(2) \\ \tilde{q}(3) \end{bmatrix}, \quad \tilde{r} = \begin{bmatrix} \tilde{r}(0) \\ \tilde{r}(1) \\ \tilde{r}(2) \end{bmatrix}.$$

The vector Γ_b is

$$\Gamma_b = \begin{bmatrix} \tilde{A}(0)\hat{X}(0) \\ \tilde{A}(1)\tilde{A}(0)\hat{X}(0) \\ \tilde{A}(2)\tilde{A}(1)\tilde{A}(0)\hat{X}(0) \end{bmatrix} = \left[\begin{array}{c} \hat{y}_f(k+1|k) \\ \hat{y}_f(k+2|k) \\ \hat{y}_f(k+3|k) \\ u(k-1) \\ \hline 0 \\ \hat{y}_f(k+2|k) \\ \hat{y}_f(k+3|k) \\ u(k-1) \\ \hline 0 \\ 0 \\ \hat{y}_f(k+3|k) \\ u(k-1) \end{array}\right]$$

The vector $\tilde{\mathscr{Q}}\Gamma_b + \tilde{q}$ is

$$\tilde{\mathscr{Q}}\Gamma_b + \tilde{q} = \begin{bmatrix} \tilde{Q}(1)\tilde{A}(0)\hat{X}(0) + \tilde{q}(1) \\ \tilde{Q}(2)\tilde{A}(1)\tilde{A}(0)\hat{X}(0) + \tilde{q}(2) \\ \tilde{Q}(3)\tilde{A}(2)\tilde{A}(1)\tilde{A}(0)\hat{X}(0) + \tilde{q}(3) \end{bmatrix} = \left[\begin{array}{c} \bar{Q}_y\hat{y}_f(k+1|k) - \bar{Q}_y r_y(k+1) \\ 0 \\ 0 \\ \bar{R}u(k-1) - \bar{r} \\ \hline 0 \\ \bar{Q}_y\hat{y}_f(k+2|k) - \bar{Q}_y r_y(k+2) \\ 0 \\ \bar{R}u(k-1) - \bar{r} \\ \hline 0 \\ 0 \\ \bar{Q}_y\hat{y}_f(k+3|k) - \bar{Q}_y r_y(k+3) \\ \bar{R}u(k-1) - \bar{r} \end{array}\right]$$

Again, each block has a number of non-zero elements that does not depend on N_p. Finally, the $\tilde{g}$ vector can be built as $\tilde{g} = \tilde{r} + \Gamma_u^T \cdot (\tilde{\mathscr{Q}}\Gamma_b + \tilde{q})$ (and taking into account the sparsity pattern of the matrices).

Overall, the cost of this condensing method is exactly the same as the condensing method applied to the original state-space system (3) and augmented to include input moves $\Delta u(k+j)$ in the problem formulation, as in Eq. (6), and equal to $\frac{1}{3}N_p^3 n_y n_u^2 + \frac{1}{3}N_p^3 n_u^3 + N_p^2 n_y^2 n_u$ flops (i.e. cubic in N_p). The condensing scheme for the *srss*-MPC is summarized in Alg. 2.

Algorithm 2 *Tailored condensing scheme for *srss*-MPC

Require: $\tilde{A}(j), \tilde{B}(j), \tilde{Q}(j), \tilde{R}(j), \tilde{q}(j), \tilde{r}(j), j = 1, \ldots, N_p$
$\hat{X}(0) := \hat{X}_f(k+1)$, and Γ_u^T {built offline}
1: create or update $\tilde{q}$, Γ_b, and $\tilde{\mathscr{Q}}\Gamma_u$
2: compute $\tilde{\mathscr{Q}}\Gamma_b + \tilde{q}$
3: $\tilde{H} \leftarrow \tilde{\mathscr{R}} + \Gamma_u^T \cdot (\tilde{\mathscr{Q}}\Gamma_u)$ {exploit fixed sparsity pattern in $\tilde{\mathscr{Q}}\Gamma_u$}
4: $\tilde{g} \leftarrow \tilde{r} + \Gamma_u^T \cdot (\tilde{\mathscr{Q}}\Gamma_b + \tilde{q})$ {exploit fixed sparsity pattern in $\tilde{\mathscr{Q}}\Gamma_b + \tilde{q}$}
5: compute $\hat{u}(j)$ {e.g. $L \leftarrow \texttt{chol}(\tilde{H})$; $\hat{u}(j) \leftarrow -L^{-1}((L^T)^{-1} + \tilde{g})$}
6: recover $\hat{X}(j)$ {i.e. simulate system dynamics using $\hat{u}(j)$}

* suitable for embedding into an IPM framework

Note that, in the current IPM framework, condensing is only a technique employed to speed up the solution of the unconstrained sub-problems producing the Newton direction. Therefore the condensing procedure is performed on-line, and its cost is added to the cost of factorizing the condensed Hessian matrix. However, the combined cost of building and factorizing the condensed Hessian compares favorably to the tailored Riccati recursion cost: if $n_u \approx n_y$, then the condensing method requires approximately $\frac{2}{3}$ the number of flops of the tailored Riccati recursion, when considering only terms cubic in N_p.

Also in this case, it is not necessary to retain all states as optimization variables in the IPM: at each stage j, only the n_y states corresponding to the output predictions $\hat{y}(k+j|k)$ and the n_u states corresponding to the inputs $u(k+j|k)$ are required.

7. Efficient step response MPC algorithms

7.1. step-*MPC*, srss-*MPC, and* ress-*MPC schemes*

This section discusses the main MPC schemes considered for step response models. Alg. 3 describes a high-level scheme that implements the *srss*-MPC, considering the proposed structure exploiting strategies and the efficient algorithms achieved for either the Riccati recursion or condensing based IPM solver.

Alg. 4 and 5 are provided in Appendix B for the *step*-MPC and *ress*-MPC schemes, respectively. Alg. 4 features the traditional step response MPC scheme that relies on the explicit prediction of future outputs. Note that, apart from the problem formulation and solution approach, the main steps involved in Alg. 3 and 4 are the same. This simply emphasizes the convenience of switching from the traditional approach to the proposed *srss*-MPC scheme.

Alg. 5 represents a viable alternative to Alg. 3 when the state-space realization algorithm (i.e. `step2ss`) is capable of producing a state-space model with state dimension $n_x \ll N_p n_y$. Note that an observer (i.e. `obs(·)`) is required for Alg. 5.

7.2. *Choice of MPC scheme and solution method*

Since Alg. 3–5 provide different MPC schemes when only step response models are available, it will be useful to have an idea of when to choose one approach over the other. First of all, due to the fact that the condensing framework achieves full exploitation of all the inherent structure in the *srss*-MPC scheme, it is easy to expect that using Alg. 2 in Alg. 3 should yield the best solver for a large range of step response MPC problem

Algorithm 3 *srss*-MPC: *step-response state-space* MPC scheme

Require: $\{\hat{y}_i(0)\}_{i=1}^{N}$, $u(k-1)$, $V(0)$, $\{S_i\}_{i=1}^{N}$, Δt (sampling interval), $\tilde{A}(j)$, $\tilde{B}(j)$, $\tilde{Q}(j)$, $\tilde{R}(j)$, $\tilde{q}(j)$, $\tilde{r}(j)$, $j=1,\ldots,N_p$ (see sections 5–6).

while CPU is running **do**
  if Δt elapsed since last call **then**
    read measurements for CVs ($y_m(k)$) and DVs (i.e. $d(k)$)
    update $V(k)$, $\{r_y(k+j), j=1,\ldots,N_p\}$
    compute $\hat{y}_f(k+j|k)$, $j=1,\ldots,N$, using e.g. Eq. (13)
    update $\tilde{X}(k)$ using $\{\hat{y}_f(k+j|k), j=1,\ldots,N_p\}$
    update $\tilde{q}(j)$ using $r_y(k+j)$
    *solve multistage problem (A.1) in a structure exploiting Riccati or condensing framework (i.e. using tailored HPMPC IPM solver based on Alg 1 or 2).
    send *optimal inputs (i.e. MVs) to plant
    shift all past data one step into the past
    update past values of variables with index $k-1$
    increment sampling time counter (i.e. $k \leftarrow k+1$)
  end if
end while

* QP problem solved to a predefined precision within Δt

Table 1: Asymptotic complexity comparison for Alg. 1 and Alg. 2, considering only the terms cubic in N_p

Solver framework	**Complexity**	**Complexity for $n_u = \alpha_1 n_y$**
Riccati (Alg. 1)	$O(N_p^3 n_y^2 n_u)$	$O(\frac{1}{\alpha_1^2} N_p^3 n_u^3)$
Condensing (Alg. 2)	$O(\frac{1}{3} N_p^3 n_y n_u^2 + \frac{1}{3} N_p^3 n_u^3)$	$O(\frac{\alpha_1+1}{3\alpha_1} N_p^3 n_u^3)$

sizes. This argument is illustrated by the asymptotic complexity comparisons made in Table 1. As mentioned at the end of Section 6.2, when $\alpha_1 = n_u/n_y = 1$, the factor of the condensing method complexity becomes $\frac{2}{3}$ (i.e. cheaper than the Riccati recursion). It is now obvious from Table 1 that the condensing strategy is cheaper whenever there are fewer inputs than outputs, and for the tailored Riccati recursion to be a better choice, the system setup should be such that $\alpha_1 = \frac{n_u}{n_y} > 1.3$, approximately.

Compared to the *ress*-MPC (Alg. 5), it is clear from the discussions in Section 6.2 that unless the realized state-space system has the same dimension as the corresponding system derived from first-principles, the condensing method for the *srss*-MPC scheme (Alg. 3) will be the best choice. When the Riccati framework is more appropriate, the *ress*-MPC may yield a faster controller. This is because the state dimension $n_x \leq N \cdot \min(n_y, n_u)$ of the realized state-space system may be less than that of the *srss*-MPC (i.e. $n_X = N_p n_y$). However, the above statement is true if $n_x \ll N_p n_y$. Recall that N is the length of the step response data sequence, and that $N_p < N$. Consider the most dominant term of the Riccati recursion in terms of n_x for the *ress*-MPC, i.e. $O(N_p n_x^3)$, and let $n_x = \alpha_2 N_p n_y$. Then, the complexity in terms of N_p and n_y becomes $O(\alpha_2^3 N_p^4 n_y^3)$. Compared to $O(\alpha_1 N_p^3 n_y^3)$ for the *srss*-MPC (see Table 1), the *ress*-MPC will be faster when $\alpha_2^3 < \frac{\alpha_1}{N_p} = \frac{n_u}{N_p n_y}$ (i.e. $n_x < (N_p^2 n_y^2 n_u)^{\frac{1}{3}}$), approximately.

In order to achieve comparable computational times for the traditional *step*-MPC (Alg. 4), compared to the tailored solution methods (Alg. 1 or 2) for the *srss*-MPC, a solver that is capable of fully exploiting the structure in the *step*-MPC will be needed. It is therefore not a good idea to use a generic solver that does not achieve a high level of sparsity exploitation for the *step*-MPC scheme.

8. Implementation in HPMPC

HPMPC is a toolbox written in ANSI C for High Performance implementation of solvers for MPC. It comes in the form of a library (i.e. not code generated), structured in layers, with special attention on code reuse.

At the bottom layer there are kernels (i.e. the innermost loop) of fundamental linear algebra routines, carefully optimized for a number of computer architectures. A key feature of these kernels is the fact that they operate on sub-matrices of the result matrix, and not on single elements: therefore there exist a 'minimum' matrix size that gives optimal performance for the linear algebra routines. For the PLC considered in this paper, this minimum size is 4×4 (see [5] for further details).

The kernels are used to implement basic linear algebra routines, optimized for the small-scale matix size typical of most embedded MPC applications. This approach combines high-performance (i.e. the kernels are optimized taking into account architecture-specific instructions and features) with portability and code reuse (i.e. the linear algebra routines implemented using these kernels are totally machine-independent). High-performance for small scale problems is also employed by means of a special matrix format that stores the matrix elements in memory, in the exact same order as they are accessed by fundamental linear algebra routines. Given the 4×4 kernels, the optimal matrix format for a matrix of size $n \times m$ is in $\frac{n}{4}$ sub-matrices (called 'panels') of $b_s = 4$ rows and m columns. This matrix format implies some limitation when operating with sub-matrices, since the linear algebra routines assume that the matrices are aligned to the top of a panel.

The linear algebra routines are used to implement solvers for unconstrained optimal control problems, as e.g. Riccati recursion or condensing methods. These in turn are used to implement solvers for constrained linear MPC, e.g. IPM or ADMM (Alternating Direction Method of Multipliers).

8.1. Implementation in a Riccati framework

The fact that HPMPC makes use of the panel-major matrix format, and that the linear algebra routines in HPMPC can only operate efficiently on sub-matrices whose top-left corner is at the top of a panel, has important consequences on the implementation choices. In particular, if n_y is not a multiple of the panel height b_s, it is not possible to implement the tailored Riccati recursion efficiently without using a reversed format of the system matrices as discussed in Section 5.2. The reversed format can be implemented efficiently, since the useful part of the matrices and vectors is always at the top-left corner.

8.2. Implementation in a condensing framework

The issues mentioned in Section 8.1 are not present in the condensing method, since the state-space formulations presented in Section 5.2 are not used directly in the solver, but only analytically in the method derivation in Section 6.2.

However, the minimum size for an optimal kernel has important practical consequences. For values of n_u smaller than b_s, the computation of $\tilde{H} = \tilde{\mathscr{R}} + \Gamma_u^T \cdot (\tilde{\mathscr{Q}}\Gamma_u)$ is inefficient when the result matrix $\tilde{H}$ is computed in sub-matrices of size $n_u \times n_u$. A better approach uses a special routine to compute the result matrix $\tilde{H}$ in sub-matrices of size 4×4, also at the cost of multiplying zero elements from Γ_u and $\tilde{\mathscr{Q}}\Gamma_u$.

9. Simulation Study

The performance of the MPC schemes discussed in this paper were verified in a simulation study, where a simple mass-damper-spring system is controlled. The system is

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \frac{1}{10}\left(\begin{bmatrix} 8 & 6 \\ -3 & 2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 4 \\ 6 \end{bmatrix} u_p(k) + \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix} d(k) \right)$$
$$y_1(k) = x_1(k) + w_1, \quad y_2(k) = x_2(k) + w_2, \quad u_p(k) = u(k) + \omega. \tag{14}$$

The system (14) is asymptotically stable, controllable and observable. A *known* disturbance variable d, which represents a persistent force, and *unknown* disturbances, affecting the plant input u_p (i.e. ω) and outputs (i.e. w_1 and w_2), are used in the simulations. The sampling time of the system is 1s, and the step response data was obtained using the `step` function in MATLAB.

9.1. Test setup

The design values used in the MPC schemes (corresponding to the definitions in problems (1) and (A.1)) are $N_p = H_p = H_u = 12$, $\underline{u} = -2.5$, $\bar{u} = 10$, $\underline{y} = [0, -10]^T$, $\bar{y} = [20, 10]^T$, $\bar{Q}_y = \mathrm{diag}(1,\ 0)$, and $\bar{P} = 1$. The penalty matrix $\bar{Q}_y$ indicates that changes in reference r_y are made for only y_1.

The state-space model (14) is used to simulate the plant, and the step response data is used in the MPC schemes. The step response data is incorporated directly into the *step*-MPC and *srss*-MPC schemes, whereas a realization routine (`step2ss`) was used to obtain a minimal state-space model from the step response data for the *ress*-MPC. The `step2ss` routine was developed based on the minimal realization approach described in [12]. Since the step response data is noise free and describes the system perfectly, the minimal realization achieved has the same size as the original system (14) (i.e. $n_x = 2$). However, the realized system has no particular structure, and the output matrix C is dense (instead of $C = I$). Consequently, the observer

$$\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{w}(k+1|k) \end{bmatrix} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \begin{bmatrix} \hat{x}(k|k-1) \\ \hat{w}(k|k-1) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} B_d \\ 0 \end{bmatrix} d(k) + \begin{bmatrix} 0 \\ I \end{bmatrix} y_m(k) \tag{15}$$

which produces both state and disturbance estimates for the *ress*-MPC was used. The matrices involved are those obtained for the minimal state-space realization:

$$A = \begin{bmatrix} 0.723 & 0.248 \\ -0.486 & 0.342 \end{bmatrix}, \ B = \begin{bmatrix} -0.840 \\ -0.582 \end{bmatrix}, \ B_d = \begin{bmatrix} -0.084 \\ -0.058 \end{bmatrix},$$
$$C = \begin{bmatrix} -0.718 & 0.428 \\ -0.234 & -0.661 \end{bmatrix}.$$

Note that the observer is deadbeat and produces exact estimates after only 1 step, and the observer gain is $[0\ I]^T$. This is in agreement with the constant disturbance model typically used in step response MPC schemes, and this particular choice is to ensure that all the tested schemes produce the same control performance. The equivalent disturbance model used in the free response computations (13) for both the *step*-MPC and *srss*-MPC schemes is $V(k) = \mathbf{1}[y_m(k) - \tilde{y}(k)]$.

For the *srss*-MPC, the tailored HPMPC solvers implementing the proposed Riccati and condensing strategies were tested, while existing generic solvers in the HPMPC toolbox were used for the *ress*-MPC. In order to obtain high performance for the traditional *step*-MPC, the CVXGEN [23] code-generated solver was used. The CVXGEN solver also implements an IPM in C code, and tailors the code to the problem data, ensuring that a high level of sparsity exploitation is achieved for the *step*-MPC. CVXGEN solves the QP problem in the form (A.2), and therefore offers a convenient framework for solving the traditional *step*-MPC problem. Due to the small size of the MPC setup, the CVXGEN code is efficient and can be optimized by the compiler to attain high-performance levels for the target platform. Similar solver parameters that lead to the same solution quality (or precision) are also available in both the CVXGEN and HPMPC solvers. Due to the above reasons, CVXGEN provides an efficient and convenient solver for the simulation study. A summary of the MPC problem setup and the properties of the controllers used are shown in Table 2.

9.2. Test platform

The test platform is the ABB AC500 PM592-ETH PLC, which has a Freescale™ G2_LE implementation of the MPC603e microprocessor, and runs at 400 MHz. The PLC has 4MB RAM for user program memory and 4MB integrated user data memory. The program code for each test controller is written in C, and it incorporates the QP solver code, according to the descriptions in Alg. 3–5. The code is compiled with `gcc 4.7.0`, `-mcpu=603e`, and optimization level set to `-O1`.

9.3. Results

A summary of the results obtained for the *step*-MPC, *srss*-MPC, and *ress*-MPC controllers are shown in Table 2. Since all the controllers solve the same MPC problem and are configured to produce the same solution quality, the same control performance was obtained for simulations using the same sequence of input and output disturbances. This is also due to the fact that perfect models and noise free step response data are used. No significant approximations are involved in the realization algorithm and the data used, and as shown in Section 5.1,

Table 2: Summary of MPC problem data, solver properties, performance results, and memory usage on the AC500 PLC

QP variant	***step*-MPC** sparse	***srss*-MPC** sparse	***srss*-MPC** dense	***ress*-MPC** sparse
Properties				
Solver	CVXGEN	HPMPC	HPMPC	HPMPC
Factorization	LDL^T	Riccati	Cholesky	Riccati
Tailoring	QP data	$A(j)$	condense[1]	none
Number of variables				
QP problem	48	48	48	72
Unconstr. sub-prob.	48	180	12	72
Performance				
Iterations[2]	9	6	6	6
Time[2] [ms]	15.69	10.27	2.24	6.88
Time/iter [ms]	1.76	1.71	0.37	1.15
Memory [MB]				
Data memory	0.048	0.059	0.032	0.066
C code size	0.17	0.15	0.19	0.15
PLC code size	0.42	0.33	0.42	0.33

[1] The structure of all matrices in the *srss*-MPC is fully exploited.
[2] Produce the same solution quality and control performance for all methods, in double precision.

the predictions used in the *step*-MPC and *srss*-MPC schemes are equivalent. This implies that the computational time of the test controllers in Table 2 can be directly compared.

The same simulation scenarios used in [15] were tested, and the performance results in Table 2 represent results that are averaged over 150 simulation time steps. As expected for interior-point methods, the worst-case time per iteration is close to the average time of each test controller. The worst-case results are therefore omitted from Table 2. The first observation is that a better computational performance is achieved by switching from the traditional *step*-MPC formulation to equivalent formulations (i.e. *srss*-MPC and *ress*-MPC) that can be solved using more efficient computational strategies. In order to obtain the same solution precision (or the same control performance), CVXGEN required more iterations to solve the *step*-MPC problem, while using similar time per iteration, compared to using the Riccati based HPMPC solver for the *srss*-MPC problem.

The results confirm that the condensing approach yields the fastest solver for the *srss*-MPC, considering the size of the MPC problem. Using the asymptotic complexity comparisons in Table 1 and the discussions in Section 7.2, a speedup factor of roughly 4 is expected, when switching from the Riccati solver to the condensing solver i.e.

$$\frac{1}{\alpha_1^2} \text{ vs. } \frac{\alpha_1+1}{3\alpha_1} \implies 4 \text{ vs. } \frac{\frac{1}{2}+1}{3\cdot\frac{1}{2}} = 1.$$

The achieved speedup factor is 4.58. Furthermore, since the state dimension of the *ress*-MPC is

$$n_{\bar{x}} = 5 < (N_p^2 n_y^2 n_u)^{\frac{1}{3}} = (12^2 \cdot 2^2 \cdot 1)^{\frac{1}{3}} \approx 8,$$

it is expected that the *ress*-MPC is faster than the *srss*-MPC when Riccati recursion is used in both controllers, cf. Section 7.2.

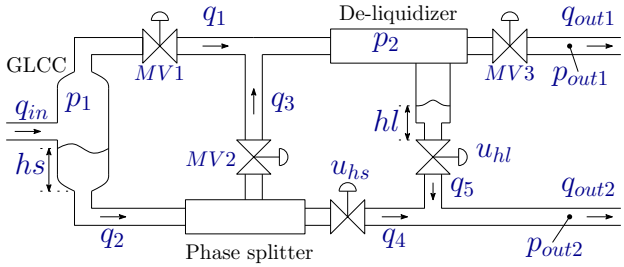

Figure 1: The subsea compact separation process

The dense *srss*-MPC requires the smallest data memory size while attaining the same code size as the CVXGEN code, generated for the *step*-MPC. However, unlike the CVXGEN code, the HPMPC solver code is not automatically generated and does not grow with increasing problem size.

10. Industrial case study

In this section, the impact of the proposed *srss*-MPC scheme and solution algorithms on an industrial example from subsea processing is discussed.

10.1. Subsea compact separator

The subsea compact separation process is well described in [24]. The process consists of separating a multiphase input flow of liquid (oil/water) and gas at two stages (see Fig. 1). First, a Gas-Liquid Cylindrical Cyclone (GLCC) separates the liquid and gas coarsely, and at the second stage a phase splitter and a de-liquidizer are used for finer separation. The main objective is to control the quality of fluid (i.e. gas volume fraction) in the gas (Gas_{out}) and liquid (Liq_{out}) outlets of the separator. It is also necessary to control two pressure variables (P_1 and P_2) around their operational points and within their safety limits, while respecting the physical limits of three control valves (labeled MV1, MV2, MV3 in Fig. 1). The valves labeled u_{hs} and u_{hl} are controlled by dedicated controllers that provide safety level control for the liquid levels h_s and h_l and ensure that MPC operates on a stable process. The variations in the liquid and gas contents of the inlet flow can be measured, and are considered as two time varying disturbances affecting the process. Due to lack of buffer volumes in the compact separator, the dynamics of the process is much faster compared to most separation techniques, and disturbance effects are much more significant. A sampling frequency of 1Hz or faster is therefore appropriate for high control performance in the presence of challenging inlet flow scenarios such as *hydrodynamic slugging*.

The process description naturally leads to a constrained multivariable control problem, and MPC is the preferred control method. Moreover, an embedded MPC solution is desirable since the compact separator is to be placed at the sea bed and has fast dynamics due to very small buffer volumes.

10.2. MPC problem setup

The MPC setup used in this case study is the same as that used in [25, 26]. Therefore, only relevant details for this paper are repeated here. Statoil's MPC tool, SEPTIC [13], was

used to obtain both noise-free step response data from a nonlinear plant simulator and MPC configurations for the controllers developed in this work. In SEPTIC, step response data is terminated when the plant response becomes approximately steady.

The MPC setup includes 4 CVs, each with 10 evaluation points. 1 CV (Liq_{out}) has an upper bound, 1 CV (Gas_{out}) has a lower bound, and 2 CVs (P_1 and P_2) have both upper and lower bounds. The bounds on the CV are considered *soft*, and therefore an 'exact penalty' method (see e.g. [8]) is used to ensure that constraint violations do not occur unless there is no feasible solution to the original problem (with *hard* bounds). Two measured DVs representing the inlet flow variations are also present. There are 3 MVs (control valves), each with 6 move blocking indices, upper and lower (hard) bounds, and (hard) bounds on the rate of change. The evaluation points and move blocks are chosen within a prediction horizon of 80.

For the *step*-MPC, this problem translates into a QP problem with 82 optimization variables, whereas the equivalent *srss*-MPC implementation has 180 variables. The difference is due to the move blocking and slack variable implementations used. In both the *step*-MPC and *srss*-MPC schemes, the CVs are evaluated at all points in the prediction horizon where control moves are allowed (except at the initial point $j = 0$, as in problem (A.1)). At CV evaluation points where MV moves are not allowed (i.e. set to zero), the corresponding input variables can be removed from the *step*-MPC. However, due to the multi-stage nature of the *srss*-MPC, even though the input moves are zero the variables are kept in the QP problem for convenience in the implementation. This accounts for 24 extra variables in the *srss*-MPC implementation, and no special strategy was employed in order to exploit the blocking information in the solution method.

In the soft constraint implementation of the *step*-MPC only 6 slack variables are used. This is due to the use of an equivalent 'exact penalty' implementation where the ∞-norm of constraint violations are penalized, instead of the 1-norm. Using the 1-norm penalty enables a straightforward implementation in the *srss*-MPC. However, this leads to a separate slack variable for each constraint at every CV evaluation point. In the HPMPC framework all constraints are considered as two sided (i.e. having both lower and upper bounds). This leads to 80 slack variables in the *srss*-MPC. Nevertheless, the extra variables rather simplify the implementation: the computational loops for the lower and upper bounds can be merged, since they have the same length, resulting in an efficient implementation.

Based on the analysis in Section 7.2, the condensing approach (Alg. 2) offers the best solver tailoring strategy for the compact separator since there are fewer MVs than CVs in the setup. When the same *minimal* realization algorithm used in Section 9 is applied to the compact separator's step response data, a rather large and dense state space model, with 184 states, is produced. The augmented state dimension of the *ress*-MPC

$$n_{\bar{x}} = (184+3) > (N_p^2 n_y^2 n_u)^{\frac{1}{3}} = (80^2 \cdot 4^2 \cdot 3)^{\frac{1}{3}} \approx 67,$$

indicates that some effort should be placed in obtaining a reduced (and acceptable) state-space realization in order to achieve a computationally fast *ress*-MPC. Unlike the *srss*-MPC that uses the CV evaluation points from the traditional step response MPC setup directly (see Section 5.5), the *ress*-MPC scheme requires much more effort to arrive at an equivalent MPC setup. The *ress*-MPC is therefore not implemented for the compact separator tests.

Table 3: Hardware-in-the-loop test results for 600 time steps of the subsea compact separation process, using the AC500 PLC.

MPC Scheme (QP Solver)	**Time** (ms) average/max	**Iterations** average/max	**Mean Square Error** $P_1/P_2/Liq_{out}/Gas_{out}$
PC-based SEPTIC MPC	–	–	0.01/0.004/3.51/0.23
srss-MPC (HPMPC[1])	17.0/20.4	10/12	0.01/0.002/3.58/0.19
step-MPC (CVXGEN[2])	68.36/81.8	15/18	0.01/0.003/2.96/0.22

[1] Implements a predictor-corrector IPM tailored according to Alg. 2.
[2] Implements a predictor-corrector IPM tailored to the problem data.

10.3. Hardware-in-the-loop test setup and results

The test setup consists of PLC implementations of the *step*-MPC and *srss*-MPC schemes, each tested in closed-loop with a nonlinear process simulator. Communication between the PLC and the simulator is achieved using Ethernet and an OPC server. The hardware-in-the-loop (HIL) test results are shown in Table 3, where the same hydrodynamic slugging flow sequence as in [25, 26] is used. In this case study, SEPTIC MPC (running on a PC) is used to produce high-performance control targets for the *step*-MPC and *srss*-MPC solutions, and the Mean Square Error values are used as the control performance measure. Double precision floating point computations were used.

The results emphasize the benefits of switching from the traditional *step*-MPC scheme to the more computationally efficient *srss*-MPC scheme, which enables the use of a more efficient structure exploiting solution method in HPMPC. As shown in Table 3, both PLC implementations achieve the high performance control targets produced by SEPTIC MPC. The computational performance is in agreement with the simulation results for the simple test system in Section 9, where both fewer iterations and faster time per iteration were achieved for the *srss*-MPC. The *srss*-MPC using HPMPC requires 1.7ms per iteration, while the CVXGEN solution for the *step*-MPC uses 4.5ms. A total speed-up of $\times 4$ is obtained by switching from the traditional approach to the *srss*-MPC scheme.

The memory usage shown in Table 4 is also in agreement with the results and discussions in Section 9. Moreover, the HPMPC based controller uses about half the PLC memory required for the CVXGEN based controller.

11. Conclusions

This paper has provided contributions to fill the gap between fast QP solver developments and industrial MPC implementations based on step response models. Different state-space realization techniques were investigated. The state-space

Table 4: Memory usage for the subsea compact separation process application on the ABB AC500 PLC.

Memory	*srss*-**MPC** (HPMPC)	*step*-**MPC** (CVXGEN)
Data memory [MB]	0.10	0.20
C code size [MB]	0.20	0.48
PLC program size [MB]	0.45	1.08

representation of step response models that relies on the recursive computation of the future output trajectory was identified as a structured model that can be exploited efficiently in a multistage QP problem.

It was shown that the explicit predictions used in the traditional step response MPC scheme translate directly to the structured recursive computations, and therefore enables the direct use of step response data in a state-space formulation. Using the recursive representation implies that the predictions in the MPC scheme can be computed in a stage-wise fashion, and therefore enables the use of block-factorization techniques in solving the resulting multistage QP problem. Based on the above observations, a novel MPC scheme is proposed (referred to as *srss*-MPC), and both a tailored Riccati recursion based IPM and a condensing based IPM are proposed for solving the *srss*-MPC problem. Since the new algorithms incorporate the original step response data in a traditional way, a dedicated state-space realization algorithm is not needed, implying that no extra model validation procedures are required in practice. However, in the likely exceptional case where a minimal state-space realization algorithm generates a state-space model of a much smaller dimension compared to that of the *srss*-MPC, the realized model will lead to a more efficient MPC controller (referred to as *ress*-MPC), when a Riccati recursion framework is used.

Asymptotic complexity analyses are used to support the discussions made in the paper, and performance results are provided for a simple MPC setup and a more complex industrial application. The results emphasize the potentials of the proposed algorithms, and it is clear that solving the proposed *srss*-MPC using the tailored condensing algorithm leads to superior solution times for a wide range of MPC problem sizes. In fact, solution times comparable to that of using the original (much smaller) state-space system is achieved for the *srss*-MPC, using the condensing scheme. The new algorithms were implemented in the HPMPC framework, and implementation aspects that lead to achieving efficient solver code for embedded MPC applications were discussed.

Appendix A. General QP problem formulations

Using Eq. (6), the MPC problem (1) can be rewritten to fit into a general multistage QP framework, assuming $k = 0$, without loss of generality:

$$\min \sum_{j=0}^{N_p-1} \left(\frac{1}{2} \begin{bmatrix} \bar{x}_j^T & \hat{u}_j^T \end{bmatrix} \begin{bmatrix} Q_j & 0 \\ 0 & R_j \end{bmatrix} \begin{bmatrix} \bar{x}_j \\ \hat{u}_j \end{bmatrix} + \begin{bmatrix} q_j^T & 0 \end{bmatrix} \begin{bmatrix} \bar{x}_j \\ \hat{u}_j \end{bmatrix} \right) + \frac{1}{2} \bar{x}_{N_p}^T Q_f \bar{x}_{N_p} \tag{A.1a}$$

$$\text{subject to} \quad \bar{x}_0 = \hat{x}, \tag{A.1b}$$

$$\bar{x}_{j+1} = \bar{A}_j \bar{x}_j + \bar{B}_j \hat{u}_j, \tag{A.1c}$$

$$\bar{E}_j \bar{x}_j + \bar{F}_j \hat{u}_j \leq \ell_j, \tag{A.1d}$$

where $j \in \{0, \ldots, N_p - 1\}$, $\hat{u}_j := \Delta u_j$, $\hat{x}$ is the current state, and for simplicity a common horizon N_p is used for both prediction and control. Definitions that can be used to obtain problem (A.1) from (1) are as follows:

$$Q_j := 2\bar{C}_j^T \bar{Q}_y \bar{C}_j, \quad Q_f := 2\bar{C}_{N_p}^T \bar{Q}_y \bar{C}_{N_p}, \quad R_j := 2\bar{P},$$

$$q_j := -\bar{C}_j^T \bar{Q}_y r_j, \quad \ell_j := \left[\bar{y}^T, \bar{u}^T, -\underline{y}^T, -\underline{u}^T, -\overline{\Delta u}^T, \underline{\Delta u}^T \right]^T,$$

$$\bar{E}_j := \left[\bar{C}_j^T, -\bar{C}_j^T, 0_{n_{\bar{x}}}, 0_{n_{\bar{x}}} \right]^T, \quad \bar{F}_j := \left[0_{n_u}, 0_{n_u}, I_{n_u}, -I_{n_u} \right]^T.$$

The reference r_j is a vector of reference states $r_x(k+j)$ and input $r_u(k+j)$, computed by a suitable setpoint filter to match the output reference $r_y(k+j)$. Note that, simply setting $Q_j = 0$ for $0 \leq j < H_w$ enforces $H_w > 1$, and it is possible to penalize the tracking error at only a few points in the prediction horizon by setting $Q_j = 0$ for the corresponding values of j. Also, it is straightforward to incorporate the constraint $\Delta u(k+j) = 0$ for $j \geq H_u$. The formulation (A.1) applies to general MIMO systems, considering block matrices and vectors of appropriate dimensions, and the system matrices $\bar{A}$, $\bar{B}$, $\bar{C}$, can vary at each stage j. A more compact form of (A.1) is:

$$\min \left\{ \tfrac{1}{2} z^T \mathbf{H} z + \mathbf{g}^T z \;\middle|\; \mathbf{A}_i z \leq \mathbf{b}_i, \;\; \mathbf{A}_e z = \mathbf{b}_e \right\}, \tag{A.2}$$

where $z = [\hat{u}_0^T, \bar{x}_1^T, \hat{u}_1^T, \ldots, \bar{x}_{N_p-1}^T, \hat{u}_{N_p-1}^T, \bar{x}_{N_p}^T]^T$, and the vectors $\mathbf{g}$, $\mathbf{b}_i$ and $\mathbf{b}_e$ as well as matrices $\mathbf{H}$, $\mathbf{A}_i$ and $\mathbf{A}_e$ are easily deduced from problem (A.1). Due to the convexity of (A.2), the Karush-Kuhn-Tucker (KKT) conditions provide both necessary and sufficient conditions for optimality.

Appendix B. Step response and realized state-space MPC

Alg. 4 and 5 summarize the requirements and the computations involved in the traditional *step*-MPC and *ress*-MPC schemes, respectively.

Acknowledgment

This work was supported by the Research Council of Norway and Statoil through the PETROMAKS project 215684.

Algorithm 4 *step*-MPC: traditional *step*-response MPC scheme

Require: $\{\hat{y}_i(0)\}_{i=1}^N$, $u(k-1)$, $V(0)$, $\{S_i\}_{i=1}^N$, Δt (sampling interval), $\mathbf{H}$, $\mathbf{g}$, $\mathbf{A}_i$, $\mathbf{b}_i$, $\mathbf{A}_e$, $\mathbf{b}_e$ according to (A.2)

while CPU is running **do**
  if Δt elapsed since last call **then**
    read measurements for CVs ($y_m(k)$) and DVs (i.e. $d(k)$)
    update $V(k)$, $\{r_y(k+j), j=1,\ldots,N_p\}$
    compute $\hat{y}_f(k+j|k)$, $j=1,\ldots,N$, using e.g. Eq. (13)
    update $\mathbf{b}_e$ using $\{\hat{y}_f(k+j|k), j=1,\ldots,N_p\}$ and $u(k-1)$
    update $\mathbf{g}$ using $r_y(k+j)$
    *solve QP problem (A.2) using a tailored solver
    send *optimal inputs (i.e. MVs) to plant
    shift all past data one step into the past
    update past values of variables with index $k-1$
    increment sampling time counter (i.e. $k \leftarrow k+1$)
  end if
end while

* QP problem solved to a predefined precision within Δt

Algorithm 5 *ress*-MPC: *realized state-space* MPC scheme

Require: $(\bar{A}, \{\bar{B}|\bar{B}_d\}, \bar{C}) = \texttt{step2ss}(\{S_i\}_{i=1}^N)$, prepared *offline*, $(\hat{x}(k+1|k), \hat{w}(k+1|k)) = \texttt{obs}(y_m(k), u(k), d(k), \hat{x}(k|k-1), \hat{w}(k|k-1))$, $\bar{x}(0) = [\hat{x}^T(0), u^T(k-1), w^T(0)]^T$, Δt (sampling interval), $Q(j)$, $R(j)$, $q(j)$, $r(j) = [r_x^T(k+j), r_u^T(k+j)]^T$, $j=1,\ldots,N_p$, according to problem (A.1).

while CPU is running **do**
  if Δt elapsed since last call **then**
    read measurements for CVs ($y_m(k)$) and DVs (i.e. $d(k)$)
    update $\{r_y(k+j), j=1,\ldots,N_p\}$
    compute $r(k+j) = [r_x^T(k+j), r_u^T(k+j)]^T$ using $r_y(k+j)$
    update $q(j)$ using $r(k+j)$
    compute current estimate for $\bar{x}(k)$ using observer, $\texttt{obs}(\cdot)$
    *solve multistage problem (A.1) in a Riccati or condensing framework (i.e. using a generic HPMPC IPM solver).
    send *optimal inputs (i.e. MVs) to plant
    update $u(k-1)$ (i.e. $u(k-1) \leftarrow u(k)$)
    increment sampling time counter (i.e. $k \leftarrow k+1$)
  end if
end while

* QP problem solved to a predefined precision within Δt

References

[1] S. J. Wright, Applying new optimization algorithms to model predictive control, in: Chemical Process Control - V, AIChE Symposium Series No. 316, Vol. 93, CACHE Publications, 1997, pp. 147–155. 1, 3

[2] C. V. Rao, S. J. Wright, J. B. Rawlings, Application of interior-point methods to model predictive control, Journal of Optimization Theory and Applications 99 (3) (1998) 723–757. 1, 3, 7

[3] Y. Wang, S. Boyd, Fast model predictive control using online optimization, IEEE Transactions on Control Systems Technology 18 (2) (2010) 267–278. 1

[4] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, C. Jones, Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control, in: IEEE Conference on Decision and Control, Maui, HI, USA, 2012, pp. 668 – 674. 1

[5] G. Frison, D. K. M. Kufoalor, L. Imsland, J. B. Jørgensen, Efficient Implementation of Solvers for Linear Model Predictive Control on Embedded Devices, in: The 2014 IEEE Multi-Conference on Systems and Control, Antibes/Nice, France, 2014. 1, 2, 10

[6] H. J. Ferreau, H. G. Bock, M. Diehl, An online active set strategy to overcome the limitations of explicit MPC, International Journal of Robust Nonlinear Control 18 (2008) 816–830. 1

[7] E. F. Camacho, C. Bordons, Model Predictive Control, Springer, 2007. 1, 2, 4, 6

[8] J. M. Maciejowski, Predictive Control: with constraints, Pearson and Prentice Hall, 2002. 1, 2, 3, 4, 6, 13

[9] J. H. Lee, M. Morari, C. E. Garcia, State-space interpretation of model predictive control, Automatica 30 (4) (1994) 707–717. 1, 4

[10] S. Li, K. Y. Lim, D. G. Fisher, A state space formulation for model predictive control, AIChE Journal 35 (2) (1989) 241–249. 1, 4

[11] B. De Schutter, Minimal state-space realization in linear system theory: an overview, Journal of Computational and Applied Mathematics 121 (1–2) (2000) 331–354. 1, 4

[12] C.-T. Chen, Linear System Theory and Design, Oxford University Press, 1999. 1, 4, 11

[13] S. Strand, J. Sagli, MPC in Statoil – advantages with in-house technology, in: Int. Symposium on Adv. Control of Chemical Processes (ADCHEM), Hong Kong, 2003, pp. 97–103. 2, 12

[14] L. Grüne, J. Pannek, Nonlinear Model Predictive Control. Theory and Algorithms., Springer, London, UK, 2011. 2

[15] D. K. M. Kufoalor, L. Imsland, T. A. Johansen, Efficient Implementation of Step response Prediction Models for Embedded Model Predictive Control, in: IFAC NMPC'15, Seville, Spain, 2015. 3, 6, 12

[16] G. Frison, J. Jørgensen, Efficient Implementation of the Riccati Recursion for Solving Linear-Quadratic Control Problems, in: 2013 IEE International Conference on Control Applications (CCA), Part of IEEE MSC 2013, Hyderabad, India, 2013, pp. 1117 – 1122. 3

[17] B. L. Ho, R. E. Kalman, Effective construction of linear state-variable models from input/output functions, in: 3rd Annual Allerton Conference on Circuit and System Theory, Monticello, Illinois, 1965, pp. 449–459. 4

[18] S. Y. Kung, A new identification and model reduction algorithm via singular value decomposition, in: 12th Asilomar Conference on Circuits, Systems and Computers, Pacific Grove, California, 1978, pp. 705–714. 4

[19] H. P. Zeiger, A. J. McEwen, Approximate linear realizations of given dimension via Ho's algorithm, IEEE Transactions on Automatic Control 19 (2) (1974) 153. 4

[20] J. B. van Helmont, A. J. J. van der Weiden, H. Anneveld, Design of optimal controllers for a coal fired Benson boiler based on a modified approximate realization algorithm, Elsevier, London, 1990, pp. 313–320. 4

[21] J. E. Ackermann, R. S. Bucy, Canonical minimal realization of a matrix of impulse response sequences, Information and Control 19 (3) (1971) 224 – 231. 4

[22] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, New York, NY, USA, 2004. 9

[23] J. Mattingley, S. Boyd, CVXGEN: A Code Generator for Embedded Convex Optimization, Optimization and Engineering 13 (1) (2012) 1–27. 11

[24] J. Høydal, O. Kristiansen, G. O. Eikrem, K. Fjalestad, Method and system for fluid separation with an integrated control system, patent nr. WO2013091719 A1 (06 2013). 12

[25] D. K. M. Kufoalor, S. Richter, L. Imsland, T. A. Johansen, M. Morari, G. O. Eikrem, Embedded Model Predictive Control on a PLC Using a Primal-Dual First-Order Method for a Subsea Separation Process, in: 22nd IEEE Mediterranean Conference on Control and Automation, Palermo, Italy, 2014. 12, 13

[26] D. K. M. Kufoalor, B. J. T. Binder, H. J. Ferreau, L. Imsland, T. A. Johansen, M. Diehl, Automatic Deployment of Industrial Embedded Model Predictive Control Using qpOASES, in: European Control Conference, Linz, Austria, 2015. 12, 13