Automatically Generated Embedded Model Predictive Control: Moving an Industrial PC-based MPC to an Embedded Platform

D. K. M. Kufoalor¹, V. Aaker¹,³, T. A. Johansen¹, L. Imsland¹, and G. O. Eikrem²

¹ Department of Engineering Cybernetics, Norwegian University of Science and Technology, O.S. Bragstads plass 2D N-7491 Trondheim, Norway.
² FD ER SF, Statoil ASA, Arkitekt Ebbells veg 10, Rotvoll, Norway.
³ V. Aaker’s current address: Aker Solutions ASA, Oslo, Norway.

SUMMARY
An embedded Model Predictive Controller (eMPC) based on the linear MPC module in SEPTIC (Statoil Estimation and Prediction Tool for Identification and Control) has been developed, and it is intended to facilitate the MPC application on embedded hardware. The control design approach illustrates a viable way of migrating from an industrial PC-based MPC technology to an embedded platform. This work also provides the transformations and the real-time considerations necessary to achieve a functional high-performance predictive controller on a low cost embedded hardware, with limited computational resources. The eMPC was implemented for rapid prototyping on the Ethernut 5 open source hardware, and its performance was tested for a simulated process. The concept was tested using a real MPC application developed for a prototype of a new industrial process that requires an advanced embedded multivariable, constraint handling control solution. The results confirm the viability of the embedded MPC design approach. Copyright © 2013 John Wiley & Sons, Ltd.

KEY WORDS: Embedded Real-time Model Predictive Control; Linear Model Predictive Control; Automatic custom code generation

1. INTRODUCTION

Model Predictive Control (MPC) is an advanced control methodology that uses multivariable process models to predict future system and control behavior. MPC aims at optimizing predicted future performance in the presence of constraints, based on the repetitive solution of an optimal control problem. Optimization in predictive control also means allowing operations closer to constraints, which in many applications, especially in the process industry, implies an increase in profitability over time.
MPC has proven to be extremely successful in numerous advanced high-level process control applications based on a software implementation in PC/server technology [1]. However, for challenging applications that require remote operation of an industrial process in an environment where the existing PC technology is not applicable, embedded computer technology such as programmable logic controllers (PLCs), field-programmable gate arrays (FPGA), digital signal processors (DSPs), and microcontrollers on ruggedized industrial form factors, become a suitable and necessary alternative. Embedded hardware could also be preferred due to power supply limitations, limited space, reliability, crucial real-time and safety requirements.

MPC is usually associated with high computational demands, making real-time implementations challenging. The computational burden becomes even more significant when dealing with embedded platforms that have limited resources. However, the challenges of embedded MPC can be met by effectively combining the power of efficient algorithms, computational performance of microprocessors, and memory management.

Recent research efforts aimed at developing efficient MPC algorithms and software for real-time and embedded applications consider either an online approach, exploiting the MPC problem structure and computational architectures [2, 3, 4, 5], or the explicit approach, which pre-computes the solution of the parameterized MPC problem offline [6, 7, 8, 9]. It is also possible to take advantage of the strengths of both online optimization and the offline computations of multi-parametric programming. This approach has led to various semi-explicit techniques [10], and also, observations from the field of parametric multi-quadratic programming are exploited in [11]. In [12, 13, 14, 15] the possibility to implement the online MPC approach on embedded hardware is explored, and some applications of the explicit approach are reported in [16, 9]. For low cost embedded devices with limited computational resources efforts towards the development of algorithms that are suitable for fixed point, instead of the widely used, but expensive, floating point computations are reported in [17, 18].

This work explores a control design approach that illustrates a viable way of migrating from an existing industrial PC-based MPC technology to an embedded hardware using portable software. In particular, an embedded model predictive controller (eMPC) based on the linear MPC module in SEPTIC (Statoil Estimation and Prediction Tool for Identification and Control [19]) has been developed. The ultimate goal of this work is to enable the use of eMPC in challenging industrial applications, and also contribute to meet the need for MPC solutions on ultra-reliable industrial embedded computers. This paper reports a step towards this objective by checking feasibility of the approach using typical hardware and optimization software. Some preliminary results were presented in [20].

The following sections cover essential aspects of the embedded MPC, including the mathematical transformations, the quadratic programming (QP) problem solver, and the real-time considerations necessary to achieve a functional high-performance predictive controller on low cost embedded hardware. To illustrate the viability of the eMPC, hardware-in-the-loop simulations are performed using an Ethernut 5 implementation of an MPC application coupled to a PC-based simulator designed for a prototype of Statoil’s compact subsea separation unit [21].
2. THE COMPACT SUBSEA SEPARATION PROCESS

The process is a separation unit that primarily separates a multiphase input flow consisting of liquid (oil/water) and gas, as illustrated in Fig. 1. The separation occurs at two stages. First, a Gas-Liquid Cylindrical Cyclone (GLCC) separates the liquid and gas coarsely, and at the second stage a Phase Splitter (PS) and a De-liquidizer (DL) are used for finer separation. The main control objectives include the control of gas volume fraction (GVF) in the gas and liquid outlets, and because the outlets lead to a compressor and a pump, it is essential that the gas and liquid contents are kept within their acceptable limits. It is also necessary to control the pressure in the GLCC and DL around their working points, as well as keeping the pressure and the pressure difference within their safety limits. The physical limits on control of all valves must also be respected. A key challenge is that, unlike most separation techniques, no buffer volumes are allowed in the subsea separation unit. Consequently, the dynamics are much faster, and disturbance effects are much more significant in the process. A good control performance is therefore required, especially for worst case flow scenarios such as hydrodynamic slugging. Slugging occurs when gas flows faster than the liquid in the inlet pipe, resulting in waves on the liquid surface that grow large enough to fill the pipe completely. Further details about the process can be found in [21]. Since the separation unit is to be placed at the seabed, it requires an embedded control strategy, and in this case an embedded MPC solution. The valves labeled $u_{hs}$ and $u_{hl}$ in Fig. 1 are controlled by dedicated controllers that provide safety level control for the liquid levels $hs$ and $hl$ and also ensure that the embedded MPC operates on a stable process. The target sampling rate is 1Hz.

3. FROM SEPTIC MPC TO EMBEDDED MPC

3.1. The design and development process

The transition from the existing SEPTIC MPC configuration to an embedded solution was accomplished through a well defined design and implementation process. Fig. 2 describes the four main stages involved. The MPC application is first designed in SEPTIC, producing a configuration (config) file and models on which the eMPC design is based. The main advantage of using SEPTIC...
(or a field-proven MPC-package) is the capability of obtaining a practically achievable control performance configuration, through the use of the numerous tuning features described in [19, 20]. The MPC can therefore be well designed and tuned to obtain the desired control performance and corresponding parameters that serve as a reliable reference and starting point for eMPC modifications.

At the second stage, a custom configuration file for the QP solver and a library-free C code for the eMPC are generated based on the SEPTIC config file and models. This is achieved by creating a C code generator written in C++. The code generator traverses the SEPTIC data structures, formulates the MPC problem based on the existing MPC application configuration and models, performs MPC problem transformations, and generates the MPC C code framework and the QP solver config file. The MPC C code framework consists of the MPC main loop from which all high level function calls are made and other supporting codes. For instance, code for disturbance estimation and calculations that update parameters used in the QP solver are generated and form part of the MPC C code framework. The solver config file is used in a custom C code generator to produce the QP solver, which is then incorporated into the generated eMPC code in the next stage.

The third stage prepares the eMPC code with required target specific code, including code for communication and the embedded platform environment initialization and setup routines. Target specific memory management code and memory allocation for large data structures are also included at this stage. The complete eMPC code is then compiled and linked using a compiler toolchain at the final stage. The final stage includes the use of extensive compiler optimization to further enhance the runtime properties of the eMPC code. For instance, a GCC compiler can be configured with CPU options that specify the architecture type, register usage, instruction scheduling parameters, and therefore allow the compiler to produce optimal code for the target CPU. Some embedded devices (e.g. Ethernut 5) offer the possibility of tailoring only necessary real-time operating system (RTOS) or firmware modules for a particular application. Customizing the RTOS and linking the required modules to the embedded application also form part of the final stage of the implementation process.

The following sections elaborate on the design and development stages.

### 3.2. The prediction model

The notations used in this section and the following sections are based on [1, 22]. The linear MPC module in SEPTIC uses mainly single-input single-output (SISO) empirical step response models, since such models are easy to build, understand, and maintain [19].
The output $y(k)$ of a sampled multiple-input multiple-output (MIMO) system can be formulated as a group of SISO step responses of a process with $n_{CV}$ controlled variables (CVs) and $n_{MV}$ manipulated variables (MVs):

$$y(k) = \sum_{i=1}^{N-1} S(i) \Delta u(k-i) + S(N)u(k-N)$$  \hspace{1cm} (1)

The variable $\Delta u$ denotes the change in input $u$, and $N$ is the number of samples (assumed equal for all SISO models). The step response matrix $S(i)$ is defined as

$$S(i) = \begin{bmatrix}
s_{1,1}(i) & s_{1,2}(i) & \cdots & s_{1,n_{MV}}(i) \\
s_{2,1}(i) & s_{2,2}(i) & \cdots & s_{2,n_{MV}}(i) \\
\vdots & \vdots & \ddots & \vdots \\
s_{n_{CV},1}(i) & s_{n_{CV},2}(i) & \cdots & s_{n_{CV},n_{MV}}(i)
\end{bmatrix}, \hspace{1cm} (2)
$$

where $s$ represents the corresponding coefficient of the step response model. The model representation (1) assumes that the step response coefficients are obtained from a process that is initially at steady-state, with all inputs and outputs at zero. It is also important to note that the step response model is valid only if the process is asymptotically stable, implying that the coefficients in $S(i)$ reach constant values after $N$ sampling periods. Otherwise, $N$ does not exist and $y(k)$ cannot be computed using (1). Although there exist a step response model generalization that covers the case where the instability is produced by integrators, low-level stabilizing controllers are mainly employed in the SEPTIC framework to ensure that the MPC operates on a stable plant. It is therefore possible to choose $N$ such that $S(N+1) \approx S(N)$.

The above MIMO system representation is based on the superposition principle, and it provides a way of generalizing linear SISO systems analysis to MIMO systems. For simplicity, the MPC problem will be first presented for SISO systems in Section 3.3, and a straightforward extension will follow for MIMO systems.

### 3.3. The MPC problem

The MPC problem can be formulated for a plant with a single output $y$ and a single control input $u$ as

$$\min_{j=H_w}^{H_p} \sum_{j=H_w}^{H_p} \|y(k+j|k) - r_y(k+j)\|_Q^2 + \sum_{j=0}^{H_u-1} \|\Delta u(k+j)\|_P^2 + \bar{\rho} \bar{\epsilon} + \rho \epsilon \hspace{1cm} (3a)$$

subject to

$$y - \epsilon \leq y(k+j|k) \leq \bar{y} + \epsilon, \quad \bar{\epsilon} \geq 0, \quad \epsilon \geq 0, \quad j \in \{H_w, \ldots, H_p\}, \hspace{1cm} (3b)$$

$$y(k+j|k) = \hat{y}(k+j|k), \quad j \in \{H_w, \ldots, H_p\}, \hspace{1cm} (3c)$$

$$\Delta u \leq \Delta u(k+j) \leq \Delta u, \quad j \in \{0, \ldots, H_u-1\}, \hspace{1cm} (3d)$$

$$u \leq u(k+j) \leq \bar{u}, \quad j \in \{0, \ldots, H_u-1\}, \hspace{1cm} (3e)$$

$$u(k+j) = u(k+j-1) + \Delta u(k+j), \quad j \in \{0, \ldots, H_u-1\}, \hspace{1cm} (3f)$$
where $k$ is the current time instant, and $k + j$ denotes the future time along the prediction horizon $H_p$ and the control horizon $H_u$. The prediction horizon starts at $H_w$, where $H_w > 1$ specifies the number of initial steps in which the deviations of the output $y(k + j|k)$ from the reference $r_y(k + j)$ are not penalized. The choice of $H_w > 1$ is particularly useful for systems where there is a time lag between the time the control action is applied and the time an effect is seen. The closed loop stability of the MPC problem can be achieved by an adequate choice of the weights, $\bar{Q} \geq 0$, $\bar{P} > 0$, and the horizon lengths, $H_p$ and $H_u$.

The MPC problem formulation uses a prediction model $\hat{y}(k + j|k)$ based on the step response models obtained from SEPTIC:

$$\hat{y}(k + j|k) = \sum_{i=1}^{j} s(i) \Delta u(k + j - i) + \sum_{i=j+1}^{N-1} s(i) \Delta \tilde{u}(k + j - i) + s(N)\tilde{u}(k + j - N) + v(k + j|k),$$

$$v(k + j|k) = v(k|k) = y_m(k) - \hat{y}(k|k - 1),$$

where the known input $\tilde{u}(\cdot)$ and input moves $\Delta \tilde{u}(\cdot)$ from the past, and the predicted input moves $\Delta \hat{u}(\cdot)$ are required. Output feedback is applied through the constant disturbance model $v(k + j|k)$, which also introduces an integral action, and thus removes steady-state control errors. The measurement of the output $y$ at time $k$ is denoted $y_m(k)$.

In (3), the slack variables $\bar{c}, \bar{\xi}$, and their weights $\bar{\rho}, \bar{\omega} > 0$, are used to relax the upper and lower constraints on the output $y(k + j|k)$, in case the optimization problem becomes infeasible. Infeasibility can occur as a result of large disturbances or prediction model errors. The use of slack variable weights as the only means of handling infeasibilities is a significant simplification compared to SEPTIC, which uses an explicit priority mechanism to further enhance the handling of infeasibilities [19, 20].

In SEPTIC, both weights on control targets and the priority of each control target (including constraints) can be assigned explicitly. A sequence of steady-state quadratic programs is therefore solved in order to respect the specified control targets (with as many of the high priority control targets as possible), and the steady-state targets are used as references for the dynamic optimization problem. The reader is referred to [19] for a good description of the main features in SEPTIC.

The next stage is to transform the MPC problem into a quadratic programming (QP) problem [20]. The approach used to derive the QP problem also includes definitions that directly extend the MPC problem (3) to MIMO systems.

### 3.4. The QP problem

The decision variables in (3) can be stacked into a vector by considering the following definitions:

$$\Delta U_j(k) = \begin{bmatrix}
\Delta u(k) \\
\Delta u(k + 1) \\
\vdots \\
\Delta u(k + H_u - 1)
\end{bmatrix}, \quad U_j(k) = \begin{bmatrix}
u(k) \\
u(k + 1) \\
\vdots \\
u(k + H_u - 1)
\end{bmatrix}, \quad Y_i(k) = \begin{bmatrix}
y(k + H_w|k) \\
y(k + H_w + 1|k) \\
\vdots \\
y(k + H_p|k)
\end{bmatrix}.$$
where the dimensions of $Q_i$ and $P_j$ are $(H_p - H_w + 1) \times (H_p - H_w + 1)$ and $H_u \times H_u$, respectively. The subscript $i = 1, 2, \ldots, n_{CV}$, and $j = 1, 2, \ldots, n_{MV}$, where $n_{CV}$ and $n_{MV}$ are the number of CVs and MVs in a MIMO system. Due to the principle of superposition, the following definitions can be used to derive a general QP for MIMO systems based on the MPC formulation introduced in Section 3.3:

$$\mathcal{T}(k) = \begin{bmatrix} T_1^T(k), & T_2^T(k), & \ldots, & T_{n_{CV}}^T(k) \end{bmatrix}^T,$$

$$Y(k) = \begin{bmatrix} Y_1^T(k), & Y_2^T(k), & \ldots, & Y_{n_{CV}}^T(k) \end{bmatrix}^T,$$

$$U(k) = \begin{bmatrix} U_1^T(k), & U_2^T(k), & \ldots, & U_{n_{MV}}^T(k) \end{bmatrix}^T,$$

$$\Delta U(k) = \begin{bmatrix} \Delta U_1^T(k), & \Delta U_2^T(k), & \ldots, & \Delta U_{n_{MV}}^T(k) \end{bmatrix}^T,$$

$$P = \text{blkdiag}(P_1, P_2, \ldots, P_{n_{MV}}), \quad \text{and}$$

$$Q = \text{blkdiag}(Q_1, Q_2, \ldots, Q_{n_{CV}}).$$

The definition of $\mathcal{T}(k)$ indicates that the setpoint can be changed at each sampling time, and it is possible to apply a setpoint trajectory produced by a setpoint optimization module. In some applications, it might be useful to define an internal reference trajectory (starting from the current output) along which the plant should be driven to the desired setpoint trajectory. The reference trajectory can be specified in such a way that a gradual transition to the desired setpoint is achieved. However, in the application considered in this paper, the plant should be driven to the setpoint as fast as possible. An internal reference trajectory is therefore not used.

Using the above definitions, the MPC problem can be rewritten as

$$\min Y(k)^T Q Y(k) + \Delta U(k)^T P \Delta U(k) - 2T(k)^T Q Y(k) + \rho_h^T \epsilon_h + \rho_1^T \epsilon_1,$$

subject to

$$GY(k) \leq g + M_h \epsilon_h + M_1 \epsilon_1, \quad \epsilon_h \geq 0, \quad \epsilon_1 \geq 0,$$

$$Y(k) = \Theta \Delta U(k) + \Psi \Delta \hat{U}(k) + \Upsilon \hat{u}(k - N) + V(k),$$

$$E \Delta U(k) \leq e,$$

$$FU(k) \leq f,$$

$$KU(k) = \Gamma \hat{u}(k - 1) + \Delta U(k),$$

where the cost function neglects the terms that do not depend on any optimization variables, and the constraints for $Y(k), U(k),$ and $\Delta U(k)$ now take a more general form. The matrices $E, F, G,$ and
their corresponding vectors \( e, f, g \), are defined as

\[
E = \text{blkdiag}(E_1, E_2, \ldots, E_{n_{MV}}), \quad e = \begin{bmatrix} e_1^T, & \ldots, & e_{n_{MV}}^T \end{bmatrix}^T,
\]

\[
F = \text{blkdiag}(F_1, F_2, \ldots, F_{n_{MV}}), \quad f = \begin{bmatrix} f_1^T, & \ldots, & f_{n_{MV}}^T \end{bmatrix}^T,
\]

\[
G = \text{blkdiag}(G_1, G_2, \ldots, G_{CV}), \quad g = \begin{bmatrix} g_1^T, & \ldots, & g_{CV}^T \end{bmatrix}^T,
\]

and the matrices involved are derived directly from the inequality constraints of (3):

\[
I_{H_u} \otimes \begin{bmatrix} 1 \\ -1 \end{bmatrix} U_j(k) \leq \begin{bmatrix} \bar{u} \\ \-\bar{u} \end{bmatrix}, \quad I_{H_u} \otimes \begin{bmatrix} 1 \\ -1 \end{bmatrix} \Delta U_j(k) \leq \begin{bmatrix} \bar{u} \\ \-\bar{u} \end{bmatrix},
\]

\[
I_{H_p - H_u + 1} \otimes \begin{bmatrix} 1 \\ -1 \end{bmatrix} Y_j(k) \leq \begin{bmatrix} \bar{y} \\ \-\bar{y} \end{bmatrix} + 1 \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \xi_k + 1 \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Omega_k,
\]

where \( \otimes \) denotes the Kronecker product, and \( \begin{bmatrix} 1 \end{bmatrix} \) is a vector with each element having a value of one. The length of \( \begin{bmatrix} 1 \end{bmatrix} \) is \( H_u \) in (7), and \( H_p - H_u + 1 \) in (8). The vectors \( \bar{m}_i, m_k \), provide the elements of the matrices \( M_h, M_l \), appearing in (6b):

\[
M_h = \text{blkdiag}(\bar{m}_1, m_2, \ldots, m_{n_y}), \quad M_l = \text{blkdiag}(m_1, m_2, \ldots, m_{n_y}).
\]

Note that the structure of \( \bar{m}_i \) and \( m_k \) represents the case where there exists both upper and lower limits on the CV. In general, the matrix \( M_h \) has 1 at corresponding entries where \( g \) represents a lower limit, and \( 0 \) at entries where \( g \) represents an upper limit, and vice versa for \( M_h \). The size of \( \epsilon_h \) and \( \epsilon_l \) corresponds to the number of CVs with high limits \( (n_y) \) and low limits \( (n_y) \), respectively, i.e.

\[
\epsilon_h = \begin{bmatrix} \epsilon_1, & \ldots, & \epsilon_{n_y} \end{bmatrix}^T, \quad \epsilon_l = \begin{bmatrix} \epsilon_1, & \ldots, & \epsilon_{n_y} \end{bmatrix}^T.
\]

The corresponding weighting values for the slack variables are also stacked in \( \rho_h \) and \( \rho_l \).

In (6f), the matrix \( K \) of size \((H_u \cdot n_{MV}) \times (H_u \cdot n_{MV})\) and matrix \( \Gamma \) of size \((H_u \cdot n_{MV}) \times n_{MV}\) result from the transformation of the relationship between \( u(k + j) \) and \( \Delta u(k + j) \) in (3f):

\[
K = \text{blkdiag}(K_1, K_2, \ldots, K_{n_{MV}}), \quad \Gamma = \text{blkdiag}(\Gamma_1, \Gamma_2, \ldots, \Gamma_{n_{MV}}),
\]

where

\[
K_j = \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 \\ -1 & 1 & \ddots & \ddots & 0 \\ 0 & -1 & \ddots & \ddots & \ddots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & -1 & 1 \end{bmatrix}, \quad \Gamma_j = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{using} \quad \tilde{u}(k - 1) = \begin{bmatrix} \tilde{u}_1(k - 1) \\ \tilde{u}_2(k - 1) \\ \vdots \\ \tilde{u}_{n_{MV}}(k - 1) \end{bmatrix}.
The following vector definitions are used in the prediction model (6c), derived from (4):

$$\Delta \tilde{U}(k) = \begin{bmatrix} \Delta \tilde{U}_1^T(k), \Delta \tilde{U}_2^T(k), \ldots, \Delta \tilde{U}_{n_{MV}}^T(k) \end{bmatrix}^T,$$

$$\Delta \tilde{U}_j(k) = \begin{bmatrix} \Delta \tilde{u}(k-1) \\ \Delta \tilde{u}(k-2) \\ \vdots \\ \Delta \tilde{u}(k+N) \\ \Delta \tilde{u}(k + H_w - N_j + 1) \end{bmatrix}, \quad \tilde{u}(k - N) = \begin{bmatrix} \tilde{u}_1(k + H_w - N_j) \\ \tilde{u}_2(k + H_w - N_j) \\ \vdots \\ \tilde{u}_{n_{MV}}(k + H_w - N_j) \end{bmatrix},$$

$$V(k) = \begin{bmatrix} 1^T v_1(k), 1^T v_2(k), \ldots, 1^T v_{n_{CV}}(k) \end{bmatrix}^T,$$

where the length of 1 is $H_p - H_w + 1$. The approach chosen to extend the MPC problem to MIMO systems allows the effects of the SISO step response sequences of different lengths to be combined in an efficient way. In contrast to the MIMO step response model (1) that uses the matrix formulation (2), $N$ is not assumed to be of the same length for all SISO models. For simplicity, the number of samples $N_j$ is defined as the largest $N$ of only the SISO step response models corresponding to MV number $j$. The remaining matrices in the prediction model (6c) are listed below.

$$\Upsilon = \begin{bmatrix} \Upsilon_{1,1}, \Upsilon_{1,2}, \ldots, \Upsilon_{1,n_{MV}} \\ \Upsilon_{2,1}, \Upsilon_{2,2}, \ldots, \Upsilon_{2,n_{MV}} \\ \vdots \quad \vdots \quad \ldots \quad \vdots \\ \Upsilon_{n_{CV},1}, \Upsilon_{n_{CV},2}, \ldots, \Upsilon_{n_{CV},n_{MV}} \end{bmatrix}_{(H_p-H_w+1)\times n_{CV}}, \quad \Upsilon_{i,j} = \begin{bmatrix} s(N_j) \\ s(N_j) \\ \vdots \\ s(N_j) \end{bmatrix}_{(H_p-H_w+1)\times 1},$$

$$\Psi = \begin{bmatrix} \Psi_{1,1}, \Psi_{1,2}, \ldots, \Psi_{1,n_{MV}} \\ \Psi_{2,1}, \Psi_{2,2}, \ldots, \Psi_{2,n_{MV}} \\ \vdots \quad \vdots \quad \ldots \quad \vdots \\ \Psi_{n_{CV},1}, \Psi_{n_{CV},2}, \ldots, \Psi_{n_{CV},n_{MV}} \end{bmatrix}_{n_{CV}(H_p-H_w+1)\times \sum_{j=1}^{n_{MV}}(N_j-H_w-1)}.$$

$$\Psi_{i,j} = \begin{bmatrix} s(H_w + 1) \quad s(H_w + 2) \quad \ldots \quad s(N_j - 2) \quad s(N_j - 1) \\ s(H_w + 2) \quad s(H_w + 3) \quad \ldots \quad s(N_j - 1) \quad s(N_j - 1) \\ \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \\ s(H_p + 1) \quad s(H_p + 2) \quad \ldots \quad s(N_j - 1) \quad s(N_j - 1) \end{bmatrix}_{(H_p-H_w+1)\times N_j-H_w-1}.$$

$$\Theta = \begin{bmatrix} \Theta_{1,1}, \Theta_{1,2}, \ldots, \Theta_{1,n_{MV}} \\ \Theta_{2,1}, \Theta_{2,2}, \ldots, \Theta_{2,n_{MV}} \\ \vdots \quad \vdots \quad \ldots \quad \vdots \\ \Theta_{n_{CV},1}, \Theta_{n_{CV},2}, \ldots, \Theta_{n_{CV},n_{MV}} \end{bmatrix}_{n_{CV}(H_p-H_w+1)\times n_{MV}\times H_u}$$. 

Copyright © 2013 John Wiley & Sons, Ltd.  
Prepared using ocaauth.cls  
DOI: 10.1002/oca
Although a simple disturbance model $V(k)$ is used to estimate the effect of unmeasured disturbances, better predictions can be achieved if information about known disturbances are used in the prediction model (6c). If a disturbance variable (DV) can be measured, a disturbance term $D(k)$ that contains the step response model of each DV-CV pair can be added to the prediction model. Since the future changes in disturbance, $\Delta d_j(k+\ell)$ for $\ell = H_w, \ldots, H_p$, are not known at the current time $k$, a usual assumption is that the future disturbances $d_j(k+\ell)$ will be the same as the current disturbance $d_j(k)$, i.e. $\Delta d_j(k+\ell) = 0$. The disturbance term can then be written as

$$D(k) = \Psi_d \Delta \tilde{D}(k-1) + \Upsilon_d \tilde{d}(k-N) + \Theta_d \Delta d(k),$$

where the matrices and vectors involved, with the exception of $\Theta_d$ and $\Delta d(k)$, are defined in a similar way as the corresponding components for the MV-CV models stated above. The matrix $\Theta_d$ can also be thought of as similar to $\Theta$ for an MV, but includes only changes at the current time, i.e.

$$\Theta_d = \begin{bmatrix}
\Theta_{d1,1}, & \Theta_{d1,2}, & \cdots, & \Theta_{d1,n_{DV}} \\
\Theta_{d2,1}, & \Theta_{d2,2}, & \cdots, & \Theta_{d2,n_{DV}} \\
\vdots & \vdots & \ddots & \vdots \\
\Theta_{dn_{CV},1}, & \Theta_{dn_{CV},2}, & \cdots, & \Theta_{dn_{CV},n_{DV}} 
\end{bmatrix}_{n_{CV}(H_p-H_w+1) \times n_{DV}}$$

$$\Delta d(k) = \begin{bmatrix}
\Delta d_1(k) \\
\Delta d_2(k) \\
\vdots \\
\Delta d_{n_{DV}}(k)
\end{bmatrix}^T$$

The coefficients that describe the step response from a DV to a CV are denoted by $s_d(\cdot)$, and $n_{DV}$ is the number of measured disturbance variables.

The final stage in the transformation process is to group the decision variables of (6) together, and the result is a standard QP:

$$\min \frac{1}{2} \begin{bmatrix}
\Delta U(k) \\
U(k) \\
Y(k) \\
\epsilon_h \\
\epsilon_l
\end{bmatrix}^T \begin{bmatrix}
2P & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2Q & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
\Delta U(k) \\
U(k) \\
Y(k) \\
\epsilon_h \\
\epsilon_l
\end{bmatrix} + \begin{bmatrix}
0 & 0 & -2T^TQ & \rho_h^T \\
0 & 0 & \rho_l^T
\end{bmatrix} \begin{bmatrix}
\Delta U(k) \\
U(k) \\
Y(k) \\
\epsilon_h \\
\epsilon_l
\end{bmatrix},$$

(9a)
subject to

\[
\begin{bmatrix}
E & 0 & 0 & 0 & 0 \\
0 & F & 0 & 0 & 0 \\
0 & 0 & G & -M_h & -M_l \\
0 & 0 & 0 & -I & 0 \\
0 & 0 & 0 & 0 & -I \\
\end{bmatrix}
\begin{bmatrix}
\Delta U(k) \\
U(k) \\
Y(k) \\
\epsilon_h \\
\epsilon_l \\
\end{bmatrix}
\leq
\begin{bmatrix}
& e \\
& f \\
& g \\
0 \\
0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-\Theta & 0 & 0 & 0 \\
-\Theta & 0 & I & 0 \\
\end{bmatrix}
\begin{bmatrix}
\Delta U(k) \\
U(k) \\
Y(k) \\
\epsilon_h \\
\epsilon_l \\
\end{bmatrix}
= \begin{bmatrix} u(k-1)\Gamma \\
\Psi \Delta \tilde{U}(k) + \Upsilon \tilde{u}(k - N) + V(k) + D(k) \end{bmatrix}
\] 

(9b)

(9c)

The QP problem (9) can be written using the compact notation:

\[
\min_z \frac{1}{2} z^T Hz + \gamma^T z \quad (H = H^T \geq 0)
\]

(10a)

subject to

\[
\bar{A}z \leq \bar{b}, \quad A_e z = b_e,
\]

(10b)

where \( z \in \mathbb{R}^n \) is the decision vector, \( b_e \in \mathbb{R}^{m_e}, \bar{b} \in \mathbb{R}^{m_i}, H \in \mathbb{R}^{n \times n} \) is a positive semi-definite Hessian, and

\[
n = 2 \cdot n_{MV} \cdot H_u + n_{CV} (H_p - H_w + 1) + n_y + n_z,
\]

\[
m_e = n_{MV} \cdot H_u + n_{CV} (H_p - H_w + 1),
\]

\[
m_i = 4 \cdot n_{MV} \cdot H_u + (n_y + n_\bar{y})(H_p - H_w + 1) + n_y + n_\bar{y}.
\]

Both parametric and constant QP data can be easily identified in the QP problem formulation, and extensive exploitation of the QP problem structure can be made during code generation. An important parameter in (9) is the equality constraint vector (i.e. \( b_e \) in (10)). This parameter contains the effect of previous input moves and disturbances on the CV predictions, and the calculation requires a matrix-vector multiplication involving the largest matrix \( \Psi \) in the QP problem. It can therefore be useful to tailor an efficient function for calculating and updating \( b_e \), independent of the QP solver used. Using a custom function for calculating \( b_e \) also allows the possibility of using different values of \( H_w \) for different CVs when necessary. The structure of \( \Psi \) assumes that \( H_w \) is the same for all CVs, else some adjustment of \( N_j \) for each CV will be necessary depending on the given application. Nevertheless, since \( H_w \) tends to be small compared to \( N_j \), the extra cost of neglecting \( H_w \) in the computation of \( b_e \) will be negligible in most cases [20]. For simplicity, \( H_w \) is omitted from \( \Psi, \Delta \tilde{U}(k), \tilde{u}(k - N) \), and the corresponding matrices used in calculating \( D(k) \) for the application considered in this paper.
For some QP solvers, a more suitable formulation can be used, where (9) is rewritten as

\[
\begin{align*}
\min_{z} & \quad \frac{1}{2}z^T H z + \gamma^T z \\
\text{subject to} & \quad A z \leq b, \quad A_e z = b_e, \quad z \in \mathbb{X},
\end{align*}
\]

(11a)

where the matrices \( H, A_e, \) and the vector \( b_e \) are obtained by directly comparing (9) and (11). The set \( \mathbb{X} \) is a convex set which is defined by all the inequalities in (9) such that the projection operator

\[
\pi_{\mathbb{X}}(z) = \arg \min_{\hat{z} \in \mathbb{X}} \frac{1}{2} \| \hat{z} - z \|^2
\]

(12)

can be evaluated analytically or by means of an algorithm with finite convergence. For MPC problem (9), set \( \mathbb{X} \) contains upper/lower bounds on \( \Delta U(k), U(k), \epsilon_h \) and \( \epsilon_l \). The bounds on \( Y(k) \) are not included in \( \mathbb{X} \) because the projection on the penalized output constraints is not simple. The remaining inequality constraints (on \( Y(k) \)) from (9) therefore define \( A \) and \( b \), i.e.

\[
A = \begin{bmatrix} 0 & 0 & G & -M_h & -M_l \end{bmatrix}, \quad b = g, \quad \text{where} \quad b \in \mathbb{R}^{(n_p+n_x)(H_p-H_w+1)}.
\]

Both QP problem formulations, (10) and (11), are sparse and can be exploited by some QP solvers. When a sparse QP formulation is most appropriate for a target QP solver, (11) is used, if required by the solver, or if the solver relies on a cheap (fast) projection (onto the constraint set \( \mathbb{X} \)) in order to boost its speed. Otherwise, (10) is used to avoid the overhead associated with any solver routine that converts (11) to (10).

An alternative approach is to use the equality constraints in (10) or (11) for elimination to obtain a more dense formulation. However, only the sparse formulations in (10) and (11) are implemented and discussed further in this paper.

4. IMPLEMENTATION ASPECTS

4.1. MV blocking and CV evaluation points

The MPC formulation and problem transformations outlined in the previous sections were extended to include problem reduction and solver speed enhancing modifications such as MV blocking and CV evaluation points. The modifications follow SEPTIC’s configuration, and introduce adequate adjustments to suit the embedded platform’s resource limitations, while at the same time meet the performance objectives of the application.

MV blocking introduces non-uniform intervals between control decisions in order to lower the computational complexity of the MPC problem [1]. Similarly, it is advantageous to design an MPC where each CV has individual prediction horizons and evaluation points [19]. Evaluation points are defined as the time instants at which the CVs are evaluated. An important note is that the MV blocking must be chosen to capture the main dynamics of the system, and such that constraint violations are avoided between CV evaluation points.
MV blocking is introduced into the cost function formulated in (3) such that MV moves are evaluated only for each MV block:

\[ \sum_{j=1}^{n_{MV}} \sum_{i=0}^{n_{Blk}-1} [\Delta u_j(\beta_j(l))]^T \bar{P}_j [\Delta u_j(\beta_j(l))], \quad \beta_j(0) = 0, \]

where an MV is allowed to change at the first time instance. The number of MVs is denoted \( n_{MV} \), \( n_{Blk} \) is the number of indexes in an MV block, and \( \beta_j(l) \) states the time index at which the \( j \)-th MV is allowed to change. Similarly, evaluation points for each CV are introduced as

\[ \sum_{i=1}^{n_{CV}} \sum_{l=1}^{n_{Ev}} [y_{ie}(\xi_i(l))]^T \bar{Q}_i [y_{ie}(\xi_i(l))], \]

where \( n_{CV} \) is the number of CVs, \( n_{Ev} \) is the number of CV evaluation points, and \( \xi_i(l) \) provides the indices of the evaluation points for the \( i \)-th CV. The deviation error is defined as \( y_{ie}(\xi_i(l)) = y_i(\xi_i(l)) - r_y(\xi_i(l)) \).

The constraints on \( y, u, \) and \( \Delta u \) also become

\[ y_i \leq y_i(\xi_i(l)) \leq \bar{y}_i, \quad l \in [1, \ldots, n_{Ev}], \]
\[ u_j \leq u_j(\beta_j(l)) \leq \bar{u}_j, \quad l \in [0, \ldots, n_{Blk} - 1], \]
\[ \Delta u_j \leq \Delta u_j(0) \leq \Delta u_j, \]
\[ (\beta_j(l) - \beta_j(l-1))\Delta u_j \leq \Delta u_j(\beta_j(l)) \leq (\beta_j(l) - \beta_j(l-1))\Delta u_j, \quad l \in [1, \ldots, n_{Blk} - 1]. \]

This reduces the number of decision variables and constraints in the QP problem derived in Section 3.4, and the expressions for the size of QP variant (10) become

\[ n = 2 \cdot n_{MV} \cdot n_{Blk} + n_{CV} \cdot n_{Ev} + n_y + n_{\bar{y}}, \]
\[ m_e = n_{MV} \cdot n_{Blk} + n_{CV} \cdot n_{Ev}, \]
\[ m_i = 4 \cdot n_{MV} \cdot n_{Blk} + n_{Ev}(n_y + n_{\bar{y}}) + n_y + n_{\bar{y}}. \]

A summary of the dimensions of the matrices involved in the resulting QP problem is presented in Table I, where \( H_w \) is omitted from the dimension of \( \Psi \) and \( \Psi_d \). In the expression for \( E \) (and \( F \)), it is assumed that both upper and lower limits exist on all MVs and MV moves, otherwise \( 2 \cdot n_{MV} \) should be replaced with the sum of upper and lower limits on MV moves (or MVs), similar to the expression for \( G \). Clearly, the sizes of the QP matrices will reduce significantly if it is possible to select much fewer points for CV evaluation and MV calculations within a long prediction horizon \( H_p \) and a long control horizon \( H_u \) in a given application.

### 4.2. The QP solver

The main approaches used to obtain fast QP solvers consider either an online strategy or an explicit strategy. The explicit approach pre-computes the solution of the parameterized MPC problem offline [6], and it is known to be most suitable for small QP problems. This paper focuses on the use of online methods in the code generation framework presented in Section 3.1, and some general
Table I. Dimensions of matrices in the reduced QP problem compared with the original QP of the form (10)

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Original QP</th>
<th>Reduced QP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q)</td>
<td>(n_{CV}(H_p - H_w + 1) \times n_{CV}(H_p - H_w + 1))</td>
<td>(n_{CV} \cdot n_{Ev} \times n_{CV} \cdot n_{Ev})</td>
</tr>
<tr>
<td>(P)</td>
<td>(n_{MV} \cdot H_a \times n_{MV} \cdot H_a)</td>
<td>(n_{MV} \cdot n_{Blk} \times n_{MV} \cdot n_{Blk})</td>
</tr>
<tr>
<td>(\Theta)</td>
<td>(n_{CV}(H_p - H_w + 1) \times n_{MV} \cdot H_a)</td>
<td>(n_{CV} \cdot n_{Ev} \times n_{MV} \cdot n_{Blk})</td>
</tr>
<tr>
<td>(\Theta_d)</td>
<td>(n_{CV}(H_p - H_w + 1) \times n_{DV})</td>
<td>(n_{CV} \cdot n_{Ev} \times n_{DV})</td>
</tr>
<tr>
<td>(\Psi)</td>
<td>(n_{CV}(H_p - H_w + 1) \times \sum_{j=1}^{n_{MV}} (N_j - 1))</td>
<td>(n_{CV} \cdot n_{Ev} \times \sum_{j=1}^{n_{MV}} (N_j - 1))</td>
</tr>
<tr>
<td>(\Psi_d)</td>
<td>(n_{CV}(H_p - H_w + 1) \times \sum_{j=1}^{n_{DV}} (N_j - 1))</td>
<td>(n_{CV} \cdot n_{Ev} \times \sum_{j=1}^{n_{DV}} (N_j - 1))</td>
</tr>
<tr>
<td>(\Upsilon)</td>
<td>(n_{CV}(H_p - H_w + 1) \times n_{MV})</td>
<td>(n_{CV} \cdot n_{Ev} \times n_{MV})</td>
</tr>
<tr>
<td>(\Upsilon_d)</td>
<td>(n_{CV}(H_p - H_w + 1) \times n_{DV})</td>
<td>(n_{CV} \cdot n_{Ev} \times n_{DV})</td>
</tr>
<tr>
<td>(E, F)</td>
<td>(2 \cdot n_{MV} \cdot H_a \times n_{MV} \cdot H_a)</td>
<td>(2 \cdot n_{MV} \cdot n_{Blk} \times n_{MV} \cdot n_{Blk})</td>
</tr>
<tr>
<td>(G)</td>
<td>((n_g + n_w)(H_p - H_w + 1) \times n_{CV}(H_p - H_w + 1))</td>
<td>((n_g + n_w)n_{Ev} \times n_{CV} \cdot n_{Ev})</td>
</tr>
<tr>
<td>(M_h)</td>
<td>((n_g + n_w)(H_p - H_w + 1) \times n_g)</td>
<td>((n_g + n_w)n_{Ev} \times n_g)</td>
</tr>
<tr>
<td>(M_l)</td>
<td>((n_g + n_w)(H_p - H_w + 1) \times n_w)</td>
<td>((n_g + n_w)n_{Ev} \times n_w)</td>
</tr>
<tr>
<td>(K)</td>
<td>(n_{MV} \cdot H_a \times n_{MV} \cdot H_a)</td>
<td>(n_{MV} \cdot n_{Blk} \times n_{MV} \cdot n_{Blk})</td>
</tr>
<tr>
<td>(\Gamma)</td>
<td>(n_{MV} \cdot H_a \times n_{MV})</td>
<td>(n_{MV} \cdot n_{Blk} \times n_{MV})</td>
</tr>
</tbody>
</table>

properties considered in selecting an appropriate QP solver for a given problem are discussed in this section. The QP solvers used in this work are presented after the general discussions, and some notes on viable alternatives that were considered, but not used, are given towards the end of this section.

Existing online methods are mainly first-order or second-order iterative methods, when classified according to the highest order of information used in finding the solution to the QP problem. First-order methods (including several variants of the fast gradient method) perform typically many, but cheap, iterations, and can be easily warm-started to attain significant computational speed-up. Warm-starting can be implemented by simply using the shifted previous solution. The sparse structure of (11) can be easily exploited by first-order method solvers. In addition, certification properties [23] and the small size of the resulting solver code make first-order methods preferable in some embedded real-time applications [24, 25, 26].

In contrast to first-order methods, second-order methods need fewer, but more expensive, iterations to converge to a solution. Second-order methods can be further grouped into interior point (IP) and active-set methods, where the iterations of active-set methods are typically more, but cheaper, than those of IP methods. Active-set methods have remarkable warm-start capabilities, whereas interior point methods require more effort for a warm-start strategy to be useful. A contribution towards solving the problems of warm-starting interior point methods is made in [27].

The sparse QP problem (10) is suitable for IP methods, while a condensed formulation is more appropriate for most active-set solvers [28]. A dense QP formulation is favorable especially when the control horizon is short.

The number of iterations is a key performance factor for both first-order and second-order methods, and the possibility of placing a predetermined hard limit on the number of iterations is a desirable property for real-time applications. Even though a QP solver may produce feasible iterates, the solutions produced when terminated too early may not possess the required accuracy for the closed loop to be stable. Real-time control guarantee will require the determination of a
lower iteration bound for which a generated input sequence is feasible, but not necessarily optimal, and satisfies performance and closed-loop stability requirements. For gradient methods and interior point methods, lower iteration bounds can be derived from their convergence rate results. However, such bounds for interior point methods are known to deviate significantly from observations in practice. In the case of active-set methods, there are no proven polynomial upper bounds on the number of iterations needed to find an optimal solution. Nevertheless, the maximum number of iterations observed in practice for active-set methods is almost always low, and if the solver cannot iterate till convergence, a strategy that provides a clear interpretation of intermediate iterates will be useful. Further discussions and examples of recent work done in this direction can be found in [29, 23, 30]. In this paper, the hard limit placed on the number of iterations for each QP solver method is determined through simulation studies.

One of the QP solvers used in this work is CVXGEN [3, 31], which uses the Mehrotra predictor-corrector interior point algorithm. CVXGEN solves two linear systems per iteration of the predictor-corrector algorithm. The linear systems are derived from the KKT conditions of a QP formulation, and the system is solved with an advanced LDLT factorization where much of the calculation of the permutation is done offline. The calculation is possible due to the knowledge of matrix sizes, sparseness properties of the QP and matrix structures. In addition, the offline preparations contribute in speeding up the online computations and reduce the complexity of the solver code. CVXGEN generates simple, flat and library-free C code suitable for the standard QP formulation in this study (no reformulations required), and therefore offers a convenient platform for direct comparison tests. The solver code generated by CVXGEN is also easily incorporated into the MPC application’s main loop outlined in section 4.3, and therefore serves as a convenient choice for this feasibility study. It is important to note that, CVXGEN works best for small problems, where the total coefficients in the constraints and objective does not exceed 2000 [31].

The dual fast gradient method and a newly proposed primal-dual first-order method [24] implemented in FiOrdOs [32, 33] were also tested for the embedded MPC code generation framework presented here. FiOrdOs is a toolbox for automated C code generation of first-order methods for the class of multi-parametric convex programs. A library-free option is also available for the generated ANSI C code. The toolbox also implements the gradient method and the fast gradient method, and offers analytical solutions where applicable. In addition, the resulting solver code, which is much less sophisticated, easier to verify, and smaller in size than the code produced by CVXGEN, is well suited for embedded applications, and therefore motivates the inclusion of the first-order QP solvers generated by FiOrdOs in this study.

The other QP solvers that were considered, but not used in the tests presented in this paper, include FORCES, which is a state-of-the-art interior point solver that generates custom library-free ANSI-C code independent of the problem size [34]. FORCES is designed to solve convex multistage problems, and it is not intended to be used for the standard QP formulation used in the MPC code generation framework presented here. The use of FORCES will therefore require the reformulation of the problem, preferably based on a state space description of the prediction model. Since this work focuses on the problem formulation presented in section 3, the FORCES solver is not used.

Another viable alternative to the QP solvers presented above is the online active set strategy of qpOASES [11]. Despite the fact that active-set methods lack convergence rate results, and usually
resort to some heuristics in the attempt to enforce iteration limits, qpOASES uses a different active-set approach that makes use of the piece-wise affine structure of the explicit solution. The special structure of QP sequences arising from MPC problems is exploited, making qpOASES particularly suited for MPC applications. The qpOASES solver is distributed as a self-contained C++ code, and because the embedded platform used in this work does not support C++ code, qpOASES is not used.

For small problems, it can be useful to examine the explicit MPC approach for real-time applications. This is mainly due to the fact that a very fast solver can be achieved using the explicit approach, and that explicit methods are well suited for determining an \textit{a priori} bound on the number of floating point operations required for a solution online. A useful tool for this purpose is the widely used Multi-Parametric Toolbox (MPT) \cite{35}, which also offers automatic C code generation.

4.3. The eMPC main loop

The complete eMPC C code described in Section 3.1 and Fig. 2 consists of the following main parts:

- The main application that starts initialization steps and runs the eMPC main loop
- A function for loading initial conditions for the eMPC
- Functions for sending and receiving data and scheduling tags
- Functions for calculating unmeasured disturbances (5)
- Functions for updating QP parameters (e.g. $b_e$ in (10) or (11))
- Functions for shifting and saving data required at the next sampling time
- The custom QP solver

The most important parts of the main loop are shown in Listing 1.

\begin{lstlisting}[language=C++]
while (1) {
    // read CV and DV measurements from plant
    readMeas();
    // calculate unmeasured disturbances
    calcUnmeasuredDisturbances();
    // update QP parameters
    updateParams();
    // solve the QP problem
    solveQP();
    // send the optimal MVs to plant
    sendData();
    // make a time shift and save data
    shiftTime();
}
\end{lstlisting}

4.4. Embedded and real-time considerations

A challenging aspect of MPC is that a numerical optimization problem must be solved at each time step. The optimization solver must therefore be fast enough to be able to produce solutions
within required real-time rates. Solving the QP problem (9) generally gives a cubic computational complexity, \(O(H_p^3 + H_w^3)\), considering the prediction and control horizons. As shown in [2] the cubic computational complexity can be reduced to linear, \(O(H_p + H_w)\), if the structure of the MPC problem is fully exploited. The computational complexity also indicates that a significant improvement in solver speed can be achieved by reducing the horizons as much as possible. Particularly, the eMPC employs MV blocking and CV evaluation points as a horizon or problem reduction strategy. The choice of evaluation points and move blocks are specified in the MPC configuration data generated by SEPTIC.

Further reduction in problem complexity is achieved through the simplified method for handling infeasibility, using slack variables with linear "exact penalty" weights in eMPC. The priority hierarchy on constraints, solved by a succession of QPs, and a quadratic cost implementation of slack variables in SEPTIC provide a better solution to solving the infeasibility problem, but at a higher computational cost compared to the simple approach used in the eMPC. As stated in the MPC formulation (3), the "exact penalty" slack variable implementation uses only one slack variable per constraint and penalizes the maximum constraint violation (as in an \(\infty\)-norm formulation). This leads to a much faster MPC compared to a quadratic or 1-norm implementation where the usual formulations use a separate slack variable for every constraint at every CV evaluation point, in order to either place a quadratic penalty on the constraint violations or penalize the sum of constraint violations.

The eMPC development stages shown in section 3.1 employ code generation, and offer a framework that incorporates custom solvers. Custom solvers can yield computation times that are several orders of magnitude faster than generic solvers, and when code generation tools are used, development times are also significantly reduced. In the existing industrial MPC software, transformation to standard form, data validity and other MPC configuration checks are performed in a parser, which calls a generic solver to compute the solution. The considered eMPC, on the other hand, uses a code generator that creates a custom solver and generates code for specific transformations. Automatic custom code generation typically yields a solver that is much faster and simpler than a parser-solver [3].

The unrolling of loops is used in the generated QP solver code [3] and in the eMPC functions used for calculating unmeasured disturbances and updating QP parameters. The technique uses a priori knowledge of loops to speed up code. For instance, offline-calculated indices can be used in large matrix operations online, instead of looping through each index in runtime. However, a trade-off with code size must be made when using this technique.

Embedded platforms are usually based on relatively simple hardware (typically a microcontroller, a PLC or an FPGA) with limited computational, memory, and data storage resources. The software implementation must therefore be kept within the runtime specifications of the embedded hardware environment. A desirable feature of a custom code generator will be to split the resulting code into an initialization part, where memory is allocated, and a real-time part, which involves no further memory allocation. This will allow a good estimate of runtime memory requirements to be determined for a given application. In addition, a generated code with few branches, allows extensive compiler code optimization to be performed, and this results in a reduced code size and runtime speedups. Another crucial implementation aspect is the conscious use of the embedded memory model of the target platform. Essentially, appropriate stack memory must be configured for
all threads. Large arrays and structures are not suitable as local variables in most cases. Such large
data structures can be placed on the target’s heap memory.

5. THE EMPC TEST PLATFORM: ETHERNUT 5

The test platform used for the eMPC test application is the open source hardware Ethernut 5 board[36]. The Ethernut 5 uses the AT91SAM9XE microcontroller, which is based on the ARM926EJ-S ARM 9 processor (CPU) that can run 200 MIPS at 180 MHz, and optimized for high internal data bandwidth. Available on the board is a 32 KB SRAM embedded in the CPU and 128 MB SDRAM for user applications. The real-time operating system (RTOS) Nut/OS is specially developed for the Ethernut board, and it offers a modular design that enables only necessary modules to be included for a given application.

A major resource limitation is the lack of a hardware floating point execution unit (FPU). Since many of the computations in the MPC application are floating point arithmetic, the number of floating point operations per second (FLOPS) is a key performance measure. In order to provide floating point support, the Nut/OS depends on the newlib [37] floating point libraries for the ARM targets.

The cross development tool used for building the Ethernut 5 application is YAGARTO [38]. It is specially developed for the ARM architecture, and it includes the GNU C/C++ toolchain. The same cross compiler and RTOS configuration settings are used to obtain all the embedded MPC test results presented in the following sections.

6. EMBEDDED MPC PERFORMANCE

6.1. Test setup and problem size

The hardware-in-the-loop simulation test setup consists of a SEPTIC process simulator (plant) running on a PC and the embedded MPC application running on the Ethernut 5 board. The embedded controller communicates with the plant through an OPC server and an Ethernet connection.

A successful real-time closed-loop test was performed on the Ethernut 5 board for the MPC application developed for a prototype of the compact separation process. The tests were done to verify and compare the control performance of the eMPC with the existing PC-based SEPTIC MPC application. In order to obtain an embedded version of the SEPTIC controller, the configuration files of the well-tuned SEPTIC MPC were used in the code generation and preparation process described in section 3.1. Apart from control target priorities and slack variable implementation, all the SEPTIC design specifications translate directly into the configuration used in the embedded controller.

The MPC application has 4 CVs with up to 10 evaluation points each, 3 MVs, each with 6 blocking indices, 2 measured process disturbances (DVs), and 6 slack variables. Since step-response models are used, the prediction horizon for each CV is calculated automatically in SEPTIC, which ensures that the modeled steady state is reached after the last MV moves. In this implementation, the MVs change at only 6 points instead of changing at every point in a horizon of 80. For each CV, one
Table II. QP problem sizes for the subsea compact separator

<table>
<thead>
<tr>
<th>QP formulation</th>
<th>Solver</th>
<th>Variables</th>
<th>Equalities</th>
<th>Inequalities</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse (10)</td>
<td>Interior point</td>
<td>82</td>
<td>58</td>
<td>138</td>
<td>−</td>
</tr>
<tr>
<td>Sparse (11)</td>
<td>First-order</td>
<td>82</td>
<td>58</td>
<td>60</td>
<td>82</td>
</tr>
</tbody>
</table>

evaluation point is placed automatically at the end of each MV block for all MVs with a model to that CV. After the last MV block, equally distributed evaluation points are configured separately for each CV. This specific choice of CV evaluation points in SEPTIC ensures that the main dynamics of the system and the essential constraint violations are captured.

The pressure in the GLCC (CV1) and the pressure in the De-liquidizer (CV2) have control setpoints and both high and low limits. The gas content of the liquid outlet (CV3) has only a high limit, and the gas content of the gas outlet (CV4) has only a low limit. Each CV limit is softened with a slack variable, whereas the limits on the MVs are implemented as hard constraints.

The eMPC test application’s real-time requirements entail solving a quadratic program within the sampling time of 1 s, using the following main matrix sizes: $Q \in \mathbb{R}^{40 \times 40}$, $P \in \mathbb{R}^{18 \times 18}$, $\rho_h \in \mathbb{R}^{1 \times 3}$, $\rho_l \in \mathbb{R}^{1 \times 3}$, $E \in \mathbb{R}^{36 \times 18}$, $G \in \mathbb{R}^{40 \times 40}$, $F \in \mathbb{R}^{36 \times 18}$, $K \in \mathbb{R}^{18 \times 18}$, $\Gamma \in \mathbb{R}^{18 \times 18}$, $\bar{\Psi} \in \mathbb{R}^{40 \times 389}$, $\bar{\Upsilon} \in \mathbb{R}^{40 \times 5}$, $\Theta \in \mathbb{R}^{40 \times 18}$, and $\Theta_d \in \mathbb{R}^{40 \times 2}$, where $\bar{\Psi} = [\Psi, \Psi_d], \bar{\Upsilon} = [\Upsilon, \Upsilon_d]$. The resulting QP problem size is summarized in Table II.

The complexity of the eMPC application is further demonstrated with some results that indicate the control efforts required to keep the CVs around their ideal values and within desired boundaries. A hydrodynamic slugging scenario is assumed in the tests, and the inlet flow sequence used consists of the two fast-changing process disturbances shown in Fig. 3.

![Inlet flow of the hydrodynamic slugging case (process disturbances)](image)

6.2. Results

The control performance of the PC-based SEPTIC MPC and a PC variant of the eMPC were tested on the same computer (PC), and the results are shown in Fig. 4 and Table III. The eMPC code used in the test setup labeled 2 in Table III is mainly the same as the code used in the tests done on the Ethernut 5 board. The only difference is the communication and platform specific parts that were adapted, compiled, and linked for the PC.
Table III. Closed-loop results for 600 time steps of the subsea compact separation process. The abbreviation \textit{sp} denotes single precision floating point, and \textit{RT} represents tests where a 1 s real-time deadline was enforced by limiting the number of iterations. For the dual fast gradient method, re-tuning was necessary.

<table>
<thead>
<tr>
<th>MPC test setup: QP Solver method</th>
<th>Time (seconds)</th>
<th>Iterations</th>
<th>Mean Square Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Platform</td>
<td>average/max</td>
<td>CV1/CV2/CV3/CV4</td>
</tr>
<tr>
<td>1: SEPTIC MPC</td>
<td>PC</td>
<td>–</td>
<td>0.01/0.004/3.51/0.23</td>
</tr>
<tr>
<td>2: Interior point</td>
<td>PC</td>
<td>0.003/0.007</td>
<td>15/17 0.04/0.008/2.38/0.29</td>
</tr>
<tr>
<td>3: Interior point, \textit{RT}</td>
<td>Ethernet 5</td>
<td>2.201/2.493</td>
<td>15/17 0.04/0.008/2.64/0.32</td>
</tr>
<tr>
<td>4: Interior point, \textit{RT}</td>
<td>Ethernet 5</td>
<td>0.982/1.000</td>
<td>6/6 0.68/0.010/0.29/1.78</td>
</tr>
<tr>
<td>5: Dual fast gradient, \textit{RT}</td>
<td>Ethernet 5</td>
<td>0.954/0.974</td>
<td>14/14 0.01/0.004/5.23/0.30</td>
</tr>
<tr>
<td>6: Dual fast gradient, \textit{RT, sp}</td>
<td>Ethernet 5</td>
<td>0.676/0.690</td>
<td>14/14 0.01/0.004/5.23/0.33</td>
</tr>
<tr>
<td>7: Primal-dual first-order</td>
<td>Ethernet 5</td>
<td>0.683/0.698</td>
<td>100/100 0.02/0.006/3.38/0.36</td>
</tr>
</tbody>
</table>

As mentioned earlier, the explicit priority specifications in SEPTIC MPC are not translated into the formulation used in the embedded MPC. The control performance achieved by the SEPTIC MPC is dictated by the following decreasing order of control target priority, specified for achieving CV setpoints and respecting constraints:

1. Respect the high limit on CV3 and the low limit on CV4
2. Respect the high and low limits on CV1
3. Achieve the setpoint target for CV1, and respect the high and low limits on CV2
4. Achieve the setpoint target for CV2

In Fig. 4, it can be seen that the SEPTIC MPC shifts its control effort to setpoint control of CV1 and CV2 whenever CV3 and CV4 are well within their bounds and close to their steady-state values. The explicit control target priority strategy implemented in SEPTIC therefore yields a slightly better control performance for the subsea compact separator in this case. Note that the plots in Fig. 4 are made large enough to make the outcome of the different control strategies obvious. From a practical point of view, the difference in control performance is rather small and insignificant. In fact, the mean square error values in the MPC test setups labeled 1 and 2 in Table III show that the control performances are close. Both controllers regulate CV1 and CV2 close to their references, while the soft constraints of CV3 (GVF in $q_{out2}$) and CV4 (GVF in $q_{out1}$) are respected satisfactorily. Moreover, a speedup of factor 20 was recorded in a comparison test, confirming the necessity and advantages of the embedded and real-time application development methods used in transforming the existing SEPTIC application into an embedded solution. Since the details of the QP solver method used in SEPTIC is not publicly available, only the control performance measures are presented in Table III. The embedded MPC performance achieved on the PC (shown as MPC test setup 2 in Table III) is therefore regarded as the target performance for the MPC running on the Ethernut 5 board. The computer used for testing the PC version of the embedded MPC has an Intel Core™ i7-3720QM processor that runs at 2.6 GHz and has 8 GB RAM. The computational time achieved for the PC version presented in Table III is intended to give an indication of the limited computational power available on the Ethernut 5 board. Even though the target embedded MPC performance was obtained using the interior point method of CVXGEN, a similar control performance on the PC can be achieved using another QP solver.
Ideally, the embedded MPC code generation and preparation framework presented in this paper should produce an embedded MPC that does not require further tuning in order to obtain a control performance close to the desired performance achieved on a PC. However, the outcome depends immensely on the optimality level achievable by the QP solver within the real-time deadline imposed on the embedded controller. It is possible to incorporate target-specific speed enhancing techniques that aim at fully utilizing the computational capacity available on the target embedded device [5], and thereby perhaps avoid relying on suboptimal solutions.
In practice, it is reasonable to choose a target embedded device that is capable of handling the computational burden of the embedded MPC application. This choice will depend on a good estimate of the upper bound on the number of iterations required in a given QP solver considering all possible operational scenarios of the plant. However, such estimates that aim at providing \textit{a priori} complexity guarantees might be conservative in practice, or even not be possible to derive for some solver methods [23]. The following sections therefore aim at verifying the real-time performance of the embedded MPC on the Ethernut 5 board, with particular attention given to the effects of the real-time computational requirements enforced on the selected QP solvers.

6.3. Results based on CVXGEN

On the Ethernut 5, an average time usage of 2.20 s was recorded when the default (strict) \textit{optimality} and \textit{numerical stability} parameters of CVXGEN’s interior point method were used. Specifically, $duality \text{ gap} = 10^{-4}$, $constraint \text{ residual} = 10^{-6}$, $maximum \text{ number of iterations} = 25$, $KKT \text{ regularization} = 10^{-7}$, and $refine \text{ steps} = 1$. The floating point computational limitations on the Ethernut board required a $hard \text{ maximum number of iterations} = 6$ to be made on the interior point solver in order to meet the desired 1 s real-time specification. This is a drastic reduction in the number of iterations considering that the optimal solution is reached at an average of 15 iterations (compare the results of MPC test setup 3 with 4 in Table III). Fig. 5 shows the control performance of the embedded interior point MPC that has reduced number of iterations, compared with the target embedded MPC performance achieved using the strict optimality parameter values.

While a relaxed specification for the $constraint \text{ residual}$ and $duality \text{ gap}$ will reduce the average solve time, the fixed iteration limit is used since the maximum solve time is more important in this case study. Note that CVXGEN uses the $regularization$ parameter to ensure that the $LDLT^T$ factorization performed when solving the KKT systems always exists. This means a perturbed system of equations is solved, and in order to recover solutions to the original system iterative refinements are used. It can therefore be expected that a careful choice of parameter combination is needed in order to obtain a stable numerical performance, and the extreme reduction of the iteration limit will produce low-quality (or even infeasible) solutions. However, despite the performance degradation observed, especially in pressure control, the reduced level of accuracy in the solutions produced does not lead to instabilities in the closed loop. Setting a hard limit on the number of iterations can therefore be considered as a means of making a trade-off between control performance and execution time for the embedded MPC based on the interior point method of CVXGEN.

The usefulness of the slack variables used to soften the constraints on the gas quality is more obvious in this case, since the controller will otherwise terminate when slugging occurs in the inlet flow.

6.4. Results based on FiOrdOs

An embedded MPC based on the dual fast gradient method (FGM) in FiOrdOs was also implemented on the Ethernut 5, and the real-time performance results are summarized in Table III, labeled as MPC test setup 4 and 5. For this method, and other first-order methods, the QP problem in the form (11) is most suitable. The dual FGM requires a positive definite Hessian, and since the original SEPTIC MPC problem leads to a positive semi-definite Hessian, the perturbed
Hessian $H_{\delta} = H + \delta \cdot I_n$, $\delta = 1 \cdot e^{-5}$ was used. The value of $\delta > 0$ was chosen to avoid producing an ill-conditioned problem.

For the compact separation MPC problem, an analytical solution of the inner problem solved in the dual FGM could be found, and the fast gradient method is used to solve the outer problem. The result is an embedded MPC that requires more than 50 iterations to produce a satisfactory control performance. The number of iterations translates to more than 2s in computational time, and therefore required the enforcement of the 1s deadline through a reduction in the iteration count. Fixing the maximum number of iterations at 14 produces solutions within the required time but...
leads to the results presented in Fig. 6. Obviously, the level of performance degradation is not

acceptable since the valve (MV2) that controls the gas outlet of the phase splitter is always closed. This control action results in a significant increase in pressure, large errors in the GLCC pressure control, and an unacceptable violation of the gas content limit in the liquid outlet of the compact separator. The MV2 was kept at zero and the consequent large violation of the gas content limit was present before the slugging of the inlet flow began after 100s. Based on this observation and the performance achieved for iterations greater than 50, any infeasibility issues that occur before the 100s mark are most likely due to the embedded controller’s own erroneous actions. When the number of iterations required to produce perturbed solutions $z^*_\delta$ close to the optimum $z^*$ of the
original problem is reduced, the observed level of performance degradation may occur since the accuracy of the solutions produced are degraded with respect to the perturbed problem.

Note that the regularization used in the interior point solver (discussed in section 6.3) also introduces perturbations to the positive semi-definite Hessian $H$. However, the solution to the perturbed system of equations obtained in each iteration of the interior point method leads to a search direction that is only a heuristic, implying that good performance can be obtained with the approximation used [3]. In addition, the iterative refinement of CVXGEN provides a strategy that moves the solution close to that of the original system.

The MPC problem solved by the dual FGM was re-tuned in an attempt to produce acceptable results using the same 14 iterations as before. The perturbation $\delta > 0$ replacing zero weights in the Hessian was no longer regarded as a single parameter, but rather as extra significant quadratic weights that introduced new tuning possibilities. The outcome is presented in Fig. 7 and in Table III, labeled as MPC test setup 4 and 5. A significant improvement is achieved, and the mean square error values clearly show the possibility of obtaining the original SEPTIC MPC performance through tuning. Moreover, the speedup obtained when single precision is used provides ample time for further performance improvement.

The results achieved using the dual FGM with only 14 iterations was possible due to the use of warm-starting and the re-tuning made. This warm starting possibility allows the number of iterations for the gradient method, which are typically many and relatively cheap (compared to the few but expensive interior point iterations of CVXGEN), to be reduced drastically. It is however worth mentioning that the observed iteration cost would have been even cheaper if all the inequality constraints, including the bounds on the outputs were simple and placed in the convex set $\mathcal{X}$ in (11). As stated at the end of section 3.4, only the bounds on $\Delta U(k), U(k), \epsilon_h$ and $\epsilon_l$ are placed in $\mathcal{X}$. Clearly, the warm-start capability of gradient methods is the main boost of speed of solution in this study, compared to interior point methods where warm-start possibilities are rather limited. Warm-starting is not supported by CVXGEN, and it is recommended on the website of CVXGEN [31] to use cold start since the performance of CVXGEN does not improve significantly with warm-start. The results obtained from the dual FGM implementation illustrate an alternative approach and the possibility of using different kinds of solvers in the embedded MPC automatic code generation framework presented here. The results are therefore not intended as a means of direct comparison of the solutions.

The capability of using the much less sophisticated first-order QP solver algorithm in controlling the complex dynamics of the compact separation process motivated further investigations leading to the introduction of a new primal-dual first-order algorithm to the control community. Promising results using this method for embedded MPC has been shown in [24]. The primal-dual first-order method does not require a positive-definite Hessian, and therefore allows direct comparison to be made with the target embedded MPC performance. The last MPC test setup (7) in Table III and Fig. 8 present the real-time performance results of the embedded primal-dual first-order MPC running on the Ethernut 5 board. Warm-starting using the shifted previous solution was implemented in this test.

Certainly, a high-speed QP solver that enables the goal of the embedded MPC code generation framework to be attained is achieved using this new method. No further tuning was necessary for the results obtained. The results give a more correct picture of what to expect from a first-order method,
in terms of iteration count for the same problem solved by the interior point method. In fact, since the dual fast gradient problem solves its inner loop analytically, it can be concluded that the overall loop count is rather expensive (compared to the iteration count of the other methods tested – see Table III). A challenging part of the MPC problem, when it comes to the first-order methods, is that the projection on the penalized output constraints is not simple. It leads to more than 30000 regions when solved parametrically with MPT [35]. The inequalities involving the outputs are therefore kept in the dual problem solved. It can then be expected that the dual FGM and the other first-order methods tested in [24] do not perform remarkably for the MPC problem setup used here.

Figure 7. Control performance of the dual fast gradient MPC after re-tuning to meet the real-time computational deadline (— —), compared with the target embedded MPC performance (——)
The final eMPC application binary file (which includes the tailored Nut/OS) for the Ethernut 5 board is 2.5MB for the first-order methods and 5.2MB for CVXGEN’s interior point method. This also presents a significant factor to consider when program memory size is a limiting resource on the targeted embedded hardware.
7. CONCLUSIONS

This research has combined custom code generation with problem size reduction methods, MPC structure preserving transformations, structure exploiting QP solvers, and other embedded real-time considerations. The feasibility of implementing an existing industrial MPC application on a typical embedded hardware has been successfully verified, and the important role played by high-speed solvers is highlighted through several hardware-in-the-loop simulation tests.

The design flexibility of well proven industrial MPC packages such as SEPTIC is combined with the academic progress in the development of efficient embedded strategies and custom high speed solvers in this paper. Hence a positive step towards bridging the developments in the industrial and academic worlds of Model Predictive Control have been made.

ACKNOWLEDGEMENT

We thank Jacob Mattingley and Stephen Boyd at Stanford University for making CVXGEN available for our academic research work. We are grateful to Fabian Ullmann, Stefan Richter, and the team of Manfred Morari at the Automatic Control Lab, ETH Zurich, for making FiOrdOs available for our research and being helpful with useful responses on questions involving the use of FiOrdOs. We also thank Stig Strand at Statoil for useful discussions on implementation aspects of SEPTIC MPC. We appreciate the detailed comments of the anonymous reviewers. Their comments and recommendations have contributed to improve this paper. The research leading to these results has benefited from collaboration within the European Union’s Seventh Framework Programme under EC-GA No. 607957 TEMPO – Training in Embedded Predictive Control and Optimization.

REFERENCES


