# Flexible Implementation of Model Predictive Control Using Sub-Optimal Solutions

Henriksson, Dan; Åkesson, Johan

2004

*Document Version:*
Publisher's PDF, also known as Version of record

ISSN 0280–5316
ISRN LUTFD2/TFRT--7610--SE

# Flexible Implementation of Model Predictive Control Using Sub-Optimal Solutions

Dan Henriksson
Johan Åkesson

Department of Automatic Control
Lund Institute of Technology
April 2004

**Department of Automatic Control**
**Lund Institute of Technology**
**Box 118**
**SE-221 00 Lund Sweden**

*Document name*
INTERNAL REPORT

*Date of issue*
April 2004

*Document Number*
ISRN LUTFD2/TFRT--7610--SE

*Author(s)*
Dan Henriksson, Johan Åkesson

*Supervisor*

*Sponsoring organisation*

*Title and subtitle*
Flexible Implementation of Model Predictive Control Using Sub-Optimal Solutions

*Abstract*

The on-line computational demands of model predictive control (MPC) often prevents its application to processes where fast sampling is necessary. This report presents a strategy for reducing the computational delay resulting from the on-line optimization inherent in many MPC formulations. Recent results have shown that feasibility, rather than optimality, is a prerequisite for stabilizing MPC algorithms, implying that premature termination of the optimization procedure may be valid, without compromising stability. The main result included in the report is a termination criterion for the on-line optimization algorithm giving rise to a sub-optimal, yet stabilizing, MPC algorithm. The termination criterion, based on an associated delay-dependent cost index, quantifies the trade-off between successively improved control profiles resulting form the optimization algorithm and the potential performance degradation due to increasing computational delay. It is also shown how the cost index may be used in a dynamic scheduling application, where the processor time is shared between two MPC tasks executing on the same CPU.

*Key words*
Model Predicive Control, Feedback Scheduling, Delay Compensation

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*
0280–5316

*ISBN*

*Language*
English

*Number of pages*
20

*Recipient's notes*

*Security classification*

# Contents


**1. Introduction** . . . . . 5

**2. MPC Formulation** . . . . . 6

2.1 Feasibility and Optimality . . . . . 7

2.2 QP-Solver . . . . . 8

**3. Termination Criterion** . . . . . 9

**4. Dynamic Real-time Scheduling of MPCs** . . . . . 11

**5. Case Study** . . . . . 12

5.1 Simulation Environment and Implementation . . . . . 12

5.2 Simulation of One MPC Controller . . . . . 13

5.3 Dynamic Scheduling of Two MPC Tasks . . . . . 15

**6. Conclusions** . . . . . 18

**7. References** . . . . . 18

# 1. Introduction

Model predictive control (MPC), see, e.g., [Garcia *et al.*, 1989; Richalet, 1993; Qin and Badgwell, 2003], has been widely accepted industrially during recent years, mainly because of its ability to handle constraints explicitly and the natural way in which it can be applied to multi-variable processes. The computational requirements of MPC, where typically a quadratic optimization problem is solved on-line in every sample, have previously prohibited its application in areas where fast sampling is required. Therefore MPC has traditionally only been applied to slow processes, mainly in the chemical industry. However, the advent of faster computers and the development of more efficient optimization algorithms, see, e.g., [Cannon *et al.*, 2001], has led to applications of MPC also to processes governed by faster dynamics. Some recent examples include [Dunbar *et al.*, 2002; Dunbar and Murray, 2002].

From a real-time implementation perspective, however, the execution time characteristics associated with MPC tasks still poses many interesting problems. Execution time measurements show that the computation time of an MPC controller varies significantly from sample to sample. The variations are due to, e.g., reference changes or external disturbances. To cope with this, an increased level of flexibility is required in the real-time implementation.

The highly varying execution times introduce delays which are hard to compensate for. The longer time spent on optimization the larger the latency, i.e., the delay between the sampling and the control signal generation. The latency has the same effect as an input time delay, and if it is not properly compensated for it will affect the control performance negatively. However, since the optimization algorithms used in MPC are iterative in nature, and, typically, reduce the quadratic cost for each iteration step, it is possible to abort the optimization before it has reached the optimum, and still fulfill the stability conditions.

Stability of model predictive control algorithms has been the topic of much research in the field. For linear systems, the stability issue is well understood, and also for nonlinear systems there are results ensuring stability under mild conditions. For an excellent review of the topic, see [Mayne *et al.*, 2000]. In summary, there are two main ingredients in most stabilizing MPC schemes; terminal penalty and terminal constraint. These two tools has been used separately or in combination to prove stability for many existing MPC algorithms. It is also well known that feasibility, rather than optimality, is sufficient to guarantee stability, see, for example [Scokaert *et al.*, 1999].

This report quantifies the control performance trade-off between successive iterations in the optimization algorithm (gradually improving the control signal quality) and the computational delay (increasing by each iteration). The trade-off is quantified by the introduction of a delay-dependent cost index, which constitutes the main contribution of this report. (A preliminary qualitative simulation study was presented in [Henriksson *et al.*, 2002a].) The index is based on a parameterization of the cost function used in the MPC formulation. The key observation is that since computational delay may significantly degrade control performance, premature termination of the optimization algorithm may be advantageous over actually finding the optimum. It will be shown how a simple termination criterion based on the cost index can be employed to improve control performance when computing resources are scarce.

Another contribution of the report, is the application of the termination criterion and cost index to real-time scheduling. Traditional real-time scheduling of control tasks is based on task models assuming constant, known, worst-case execution times for all tasks. However, the large variations in execution time for MPC tasks render real-time designs based on worst-case bounds very conservative and give unnecessary long sampling periods. Hence, more flexible implementation schemes than traditional fixed-priority or deadline-based scheduling are needed.

In feedback scheduling, [Årzén *et al.*, 2000; Cervin *et al.*, 2002], the CPU time is viewed as a resource that is distributed dynamically between the different tasks based on, e.g., feedback from CPU usage and quality-of-service (QoS). For controller tasks the quality-of-service corresponds to the control performance. Another approach that can be tailored towards MPC is scheduling of imprecise computations [Liu *et al.*, 1991; Liu *et al.*, 1994]. Here, each task is divided in a mandatory part (finding a feasible solution) and an optional part (QP optimization), which are scheduled separately. The dynamic scheduling strategy proposed in this report schedules the optional parts of the MPC tasks using the cost indices as dynamic task priorities.

The rest of the report is organized as follows. The MPC formulation is given in Section 2. Section 3 describes a termination criterion, based on the delay-dependent cost index, which is used to dynamically trade-off computational delay and optimization. Section 4 describes a dynamic scheduling scheme for scheduling of multiple MPC controllers. Section 5 contains a case study, evaluating the performance of the termination strategy and dynamic scheduling scheme. Finally the conclusions are given in Section 6.

## 2. MPC Formulation

The MPC formulation is based on [Maciejowski, 2002] and assumes a discrete, linear process model on the form

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= C_y x(k) \\ z(k) &= C_z x(k) + D_z u(k) \end{aligned} \tag{1}$$

where $y(k)$ is the measured output, $z(k)$ the controlled output, $x(k)$ the state vector, and $u(k)$ the input vector. The function to minimize at time $k$ is

$$J(k, \Delta u, x(k)) = \sum_{i=1}^{H_p} \|\hat{z}(k+i|k) - r(k+i)\|_Q^2 + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i|k)\|_R^2 \tag{2}$$

where $\hat{z}$ is the predicted controlled output, $r$ is the current set-point, $\hat{u}$ is the predicted control signal, $H_p$ is the prediction horizon, $H_u$ is the control horizon, $Q \geq 0$ and $R > 0$ are weighting matrices, and $\Delta u(k) = u(k) - u(k-1)$. It is assumed that $H_u < H_p$ and that $\hat{u}(k+i) = \hat{u}(k+H_u-1)$ for $i \geq H_u$. See Figure 1. $\Delta u = \left(\Delta\hat{u}(k)^T \dots \Delta\hat{u}(k+H_u-1)^T\right)^T$ is the solution vector.

Introducing sequences $u$ and $z$ equivalently to $\Delta u$, the state and control signal constraints may be expressed as

$$W\Delta u \leq w \quad Fu \leq f \quad Gz \leq g \tag{3}$$

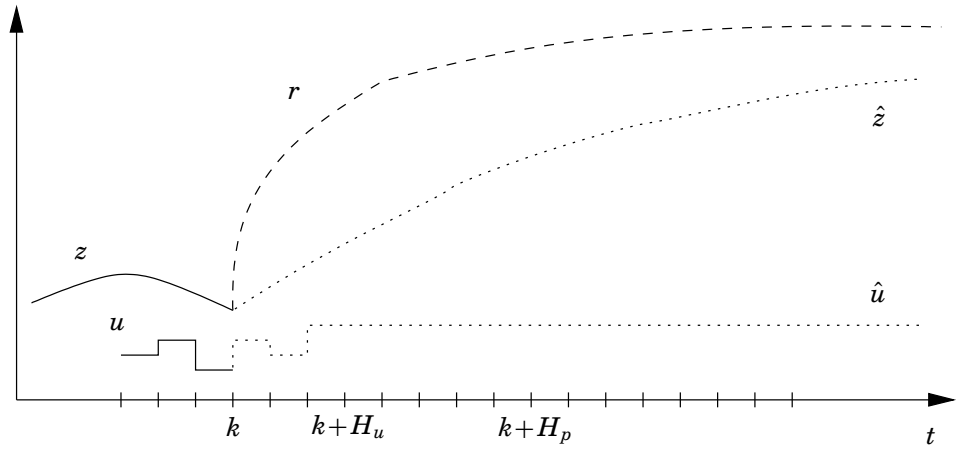


**Figure 1** The basic principle of model predictive control.

This formulation leads to a convex linear-inequality constrained quadratic programming problem (LICQP) to be solved at each sample. The problem can be written on matrix form as

$$\min_{\theta} \; V(k) = \theta^T \mathcal{H} \, \theta - \theta^T \mathcal{G} \; + \mathcal{C} \quad \text{s.t.} \quad \Omega\theta \leq \omega. \tag{4}$$

where $\theta = \Delta\mathcal{U}$ and the matrices $\mathcal{H}$ , $\mathcal{G}$ , $\mathcal{C}$ , $\Omega$, and $\omega$ depend on the process model and the constraints, see [Maciejowski, 2002]. Only the first element of $\Delta\mathcal{U}$ is applied to the process and the optimization is then repeated in the next sample. This is referred to as the *receding horizon* principle, see Figure 1.

### 2.1 Feasibility and Optimality

The problem of formulating stabilizing MPC schemes has received much attention in the last decade. For linear MPC, the conditions for stability are well understood, and several techniques for ensuring stability exist, including terminal penalty, terminal equality constraint, and terminal sets, see [Mayne *et al.*, 2000]. For simplicity, we will use a terminal equality constraint to ensure stability, see, e.g., [Bemporad *et al.*, 1994].

The following theorem (adopted from [Bemporad *et al.*, 1994]) summarizes the important features of a stabilizing MPC scheme based on a terminal equality constraint. Without lack of generality we assume that $r(k)$ is zero.

THEOREM 1
Consider the system (1) controlled by the receding horizon controller based on the cost function (2), subject to the constraints (3). Let $r(k)$=0. Further assume terminal constraints $\hat{x}(k+H_p+1)$=0 and $\hat{u}(k+H_u)$=0, $Q \geq 0$ and $R>0$ and that $(Q^{\frac{1}{2}}C_z, A)$ is a detectable pair. If the optimization problem is feasible at time $k$, then the origin is stable, and $z(k)^T Q z(k) \to 0$ as $k \to \infty$.

*Proof.* Let $\Delta\mathcal{U}_k^*=(\Delta\hat{u}_k^*(k), \Delta\hat{u}_k^*(k+1), \ldots, \Delta\hat{u}_k^*(k+H_u-1))$ denote the optimal control sequence at time $k$. Obviously, $\Delta\mathcal{U}_{k+1}=(\Delta\hat{u}_k^*(k+1), \ldots, \Delta u_k^*(k+H_u-1),\ 0)$ is then feasible at time $k+1$. Consider the function $V(k)=J(k, \Delta\mathcal{U}_k^*, x(k))$ with

$r(k)$=0. Then we have the following relations:

$$\begin{aligned} V(k+1) &= J(k+1, \Delta u^{*}_{k+1}, x(k+1)) \\ &\leq J(k+1, \Delta u_{k+1}, x(k+1)) \\ &= V(k) - z(k+1)^T Q z(k+1) \\ -\Delta u(k)^T R \Delta u(k). & \end{aligned} \tag{5}$$

Since $V(k)$ is lower-bounded and decreasing, $z(k)^T Q z(k) \to 0$ and $\Delta u(k)^T R \Delta u(k) \to 0$ as $k \to \infty$. Further, using the fact that $(Q^{\frac{1}{2}} C_z, A)$ is a detectable pair, it follows that $\|x(k)\| \to C < \infty$ as $k \to \infty$. □

REMARK 1
To prove the stronger result that the origin is asymptotically stable, the additional assumption that the system (1) has no transmission zeros at $q = 1$ from $u$ to $z$ could be imposed. Notice also that the sensible assumption that $Q > 0$ implies that $z(k) \to 0$ as $k \to \infty$, which is, however, automatically achieved if the transmission zero condition is fulfilled. □

The important feature in the proof of this theorem is embedded in equation (5). In order for the stability proof to work, it must be ensured that $V(k)$ is decreasing, which, however, does not require optimality of the control sequence $\Delta u$ . See, e.g., [Scokaert *et al.*, 1999] for a thorough discussion on this topic. Rather, having fulfilled the stability condition $V(k+1) < V(k)$, the optimization may be aborted prematurely without losing stability. In the case study in Section 5, the terminal constraint $\hat{u}(k + H_u)$=0 has been relaxed, in order to increase the feasibility region of the controller. To remove this complication, the control signal, $u$, rather than the control increments, $\Delta u$, could be included in the cost function. Notice, however, that the important feature of the stability proof that will be explored is the inequality (5) and that other, more sophisticated, stabilizing techniques may well be used instead.

### 2.2 QP-Solver

There are two major families of algorithms for solving LICQPs; *active set methods* [Fletcher, 1991] and *primal-dual interior point methods*, e.g., Mehrotra's predictor-corrector algorithm, [Wright, 1997]. Both types of methods have advantages and disadvantages when applied to MPC, as noted in [Bartlett *et al.*, 2000] and [Maciejowski, 2002]. Rather, the key to efficient algorithms lies in exploring the structure of the MPC optimization problem.

Recent research has also suggested interesting, and fundamentally different MPC algorithms, see, e.g., [Kouvaritakis *et al.*, 2002] and [Bemporad *et al.*, 2002], known as explicit MPC. Here, the optimization problem is solved *off-line* for all $x(k)$, resulting in an explicit piecewise affine control law. At run-time, the problem is then transformed into finding the appropriate (linear) control law, based on the current state estimation. However, when the complexity of the problem increases, so does the complexity of the problem of finding the appropriate control law at each sample.

An MPC algorithm based on the on-line solution of a QP-problem is used in this report. The value of the cost function at each iteration in the optimization algorithm is of importance. Specifically, if the decay of the cost function is slow, it may be a good choice to terminate the optimization algorithm, and use

the sub-optimal solution, rather than allowing the algorithm to continue and thereby introduce additional delay in the control loop. In the scheduling case, long execution times will also affect the performance of other control loops.

From this point of view, there is a fundamental difference between an active set algorithm and a typical primal-dual interior point method. The active set algorithm explicitly strives to decrease the cost function in each iteration, whereas a primal-dual interior point algorithm rather tries to find, simultaneously, a point in the primal-dual space that fulfills the Karush-Kuhn-Tucker conditions. In the latter case, the duality gap is explicitly minimized in each iteration, rather than the cost function. With these arguments, and from our experience using both types of algorithms, we conclude that an active set algorithm is preferable for the application in this report.

# 3. Termination Criterion

To be able to determine when to abort the MPC optimization and output the control signal, it is necessary to quantify the trade-off between the performance gain resulting from subsequent solutions of the QP-problem, and the performance loss resulting from the added computational delay. This will be achieved by the introduction of a delay-dependent cost index, which is based on a parameterization of the cost function (2).

Assuming a constant time delay, $\tau < h$, the process model (1) can be augmented (see, e.g., [Åström and Wittenmark, 1997]) as

$$\begin{aligned} \tilde{x}(k+1) &= \tilde{\Phi}\tilde{x}(k) + \tilde{\Gamma}u(k) \\ y(k) &= \tilde{C}_y\tilde{x}(k) \\ z(k) &= \tilde{C}_z\tilde{x}(k) + D_z u(k) \end{aligned} \tag{6}$$

where

$$\begin{aligned} \tilde{x}(k) &= \begin{pmatrix} x(k) & u(k-1) \end{pmatrix}^T \\ \tilde{\Phi} &= \begin{pmatrix} \Phi & \Gamma_1(\tau) \\ 0 & 0 \end{pmatrix}, \qquad \tilde{\Gamma} = \begin{pmatrix} \Gamma_0(\tau) \\ 1 \end{pmatrix} \\ \tilde{C}_y &= \begin{pmatrix} C_y & 0 \end{pmatrix}, \qquad \tilde{C}_z = \begin{pmatrix} C_z & 0 \end{pmatrix} \\ \Gamma_0(\tau) &= \int_0^{h-\tau} e^{As} ds\ B, \quad \Gamma_1(\tau) = e^{A(h-\tau)} \int_0^{\tau} e^{As} ds\ B \end{aligned}$$

and $A$ and $B$ are the corresponding continuous time system matrices of the plant. The matrices $\mathcal{H}$, $\mathcal{G}$, $\mathcal{C}$, $\Omega$, and $\omega$ in (4) all depend on the system matrices and thus on the delay, $\tau$. Ideally, these matrices should be updated from sample to sample based on the current computational delay.

However, using the representation (6) it is possible to evaluate the cost function (2) assuming a constant computational delay, $\tau$, over the prediction horizon. The assumption that the delay is constant over the prediction horizon is in line with the assumptions commonly made in the standard MPC formulation, e.g., that the current reference values will be constant over the prediction horizon. Thus, for each iterate, $\Delta\mathcal{U}_i$, produced by the optimization algorithm, we compute

$$J_d(\Delta\mathcal{U}_i, \tau) = \Delta\mathcal{U}_i^T \mathcal{H}(\tau)\Delta\mathcal{U}_i - \Delta\mathcal{U}_i^T \mathcal{G}(\tau) + \mathcal{C}(\tau) \tag{7}$$

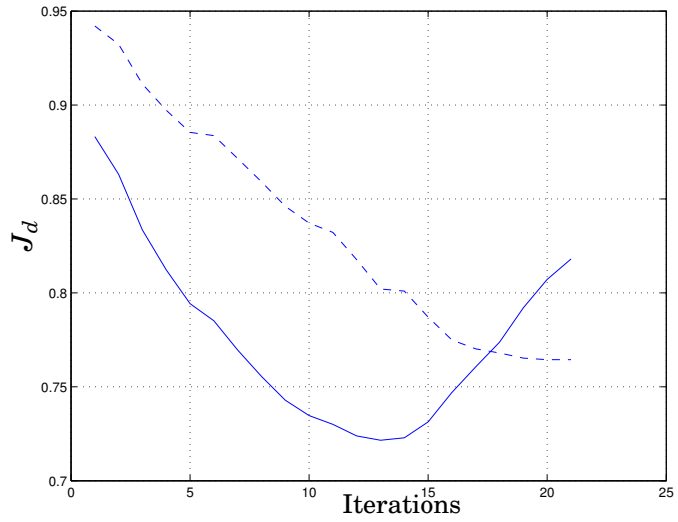


**Figure 2** The solid curve shows the delay-dependent cost index $J_d$, and the dashed curve shows the original cost function used in the QP-algorithm.

This cost index penalizes not only deviations from the desired reference trajectory, but also performance degradation due to the current computational delay, $\tau$. There are two major factors that affect the evolution of $J_d$. On one hand, an increasing $\tau$, corresponding to an increased computational delay, may degrade control performance and cause $J_d$ to increase. On the other hand, $J_d$ will decrease for successive $\Delta u_i$:s since the quality of the control signal has improved from the last iteration. Figure 2 shows the evolution of $J_d$ during an optimization run. In the beginning of the optimization, $J_d$ is decreasing rapidly, but then increases due to computational delay. In this particular example, the delayed control trajectory seems to achieve a lower cost than the original. This situation may occur since the cost functions are evaluated for non-optimal control sequences, except for the last iteration. Notice, however, that for the optimal solution, $J_d$ is higher than the original cost. The proposed termination strategy is then to compare the value of $J_d(\Delta u_i, \tau_i)$ with the cost index computed after the previous iteration, i.e., $J_d(\Delta u_{i-1}, \tau_{i-1})$, where $\tau_i$ denotes the current computational delay after the $i$th iteration. If the cost index has decreased since the last iteration, we conclude that we gained more by optimization than we lost by the additional delay. On the other hand, if the cost index has increased, the optimization may be aborted. However, the requirement stemming from the stability proof, i.e., $V(k+1) \leq V(k)$ must also be fulfilled if the optimization algorithm is to be terminated prematurely. Notice that the matrices needed to evaluate $J_d$ should be calculated off-line.

The MPC formulation assumes a process model without delay. Another possible approach would be to include a fixed-sample delay in the process description. However, since the computational delay is highly varying, compensating for the maximum delay may become very pessimistic and lead to decreased obtainable performance. We will also assume that the control signal is actuated as soon as the optimization algorithm terminates, not to induce any unnecessary delay.

# 4. Dynamic Real-time Scheduling of MPCs

The cost index and termination criterion described above, will now be applied in a dynamic real-time scheduling context. Controller tasks are often implemented as tasks on a microprocessor using a real-time kernel or a real-time operating system (RTOS). The real-time kernel or OS uses multiprogramming to multiplex the execution of the tasks on the CPU. To guarantee that the time requirements and time constraints of the individual tasks are all met, it is necessary to schedule the usage of the CPU time.

During the last two decades, scheduling of CPU time has been a very active research area and a number of different scheduling models and methods have been developed [Buttazzo, 1997; Liu, 2000]. The most common, and simplest, model assumes that the tasks are periodic, or can be transformed to periodic tasks, with a fixed period, $T_i$, a known worst-case execution time, $C_i$, and a *hard deadline*, $D_i$. The latter implies that it is imperative that the tasks always meet their deadlines, i.e., that the actual execution time in each sample is always less or equal to the deadline.

MPC tasks, however, do not fit this traditional task model very well, mainly because their highly varying execution times. On the other hand, MPC offers two features that distinguish it from ordinary control algorithms from a real-time scheduling perspective. First, as we have seen in the previous section, it is possible to abort the computation and thereby reduce the execution time. Second, the cost index contains relevant information about the state of the controlled process. Thus, the cost index can be viewed as a real-world quality-of-service measure for the controller, and be used as a dynamic task priority by the scheduler. This also enables a tight and natural connection between the control and the real-time scheduling.

The MPC algorithm can be divided into two parts. The first part consists of finding a starting point fulfilling the constraints in the MPC formulation (constraints on the controlled and control variables and the terminal equality constraint) and to iterate the QP optimization algorithm until the stability condition of Theorem 1 is fulfilled. The second part consists of the additional QP iterations that further reduce the value of the cost function. The second part of the algorithm may be aborted without jeopardizing stability, as discussed above.

Based on this insight, the MPC algorithm can be cast into the framework of scheduling of imprecise computations [Liu *et al.*, 1991; Liu *et al.*, 1994]. Using their terminology, the first part of the MPC algorithm is called the mandatory sub-task, and the second part is called the optional sub-task. The mandatory sub-tasks will be given the highest priority, whereas the optional sub-tasks will be scheduled based on the values of the MPC cost indices. Listing 1 contains pseudo code of a dynamic scheduling scheme of the optional sub-tasks. The strategy also exploits the trade-off between optimization and computational delay.

It should be noted that comparing cost indices directly may not be appropriate when the controllers have different sampling intervals, prediction horizons, weighting matrices, etc. In those cases, it would be necessary to scale the cost indices to obtain a fair comparison. The scheduling could also use feedback from the derivatives of the cost functions, as well as the relative deadlines of the different controllers.

**Listing 1** Dynamic real-time scheduling strategy for MPC tasks.

```
determine MPC sub-task i with highest J_d;
schedule sub-task i for one iteration;

now = currentTime;
delay_i = now - start_i;

if (optimum_reached_i) {
   actuate plant_i;
   oldcost_i = J_d(u_i, delay_i);

} else {
   costIndex = J_d(u_i,delay_i);
   costIndexInc == (costIndex > oldCostIndex);
   stabilityReq == (costIndex < oldcost_i);

   if (costIndexInc && stabilityReq) {
      abort optimization;
      actuate plant_i;
      oldcost_i = J_d(u_i, delay_i);
   }
   oldCostIndex = costIndex;
}
```

# 5. Case Study

The proposed termination criterion and dynamic real-time scheduling strategy have been evaluated in simulation using a second order system, a double-integrator:

$$\begin{aligned} \dot{x} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \\ y &= \begin{pmatrix} 1 & 0 \end{pmatrix} x \end{aligned} \tag{8}$$

The plant was discretized using the sampling interval $h = 0.1$ s. In the simulations, $z = x_1$ was set to be the controlled state and the constraints $|u| \leq 0.3$ and $|x_2| \leq 0.1$ were enforced. The MPC controller was implemented as described in Section 2, with prediction horizons $H_p = 50$ and $H_u = 20$ and weighting matrices $Q = 1$ and $R = 0.1$.

## 5.1 Simulation Environment and Implementation

Real-time MPC control of the double-integrator process was simulated using the TrueTime toolbox [Henriksson *et al.*, 2002b]. Using TrueTime it is possible to perform detailed co-simulation of the MPC control task executing in a real-time kernel and the continuous dynamics of the controlled process. Using the toolbox it is easy to simulate different implementation and scheduling strategies and evaluate them from a control performance perspective.

In the standard implementation, the MPC task is released periodically and new instances may not start to execute until the previous instance has completed.

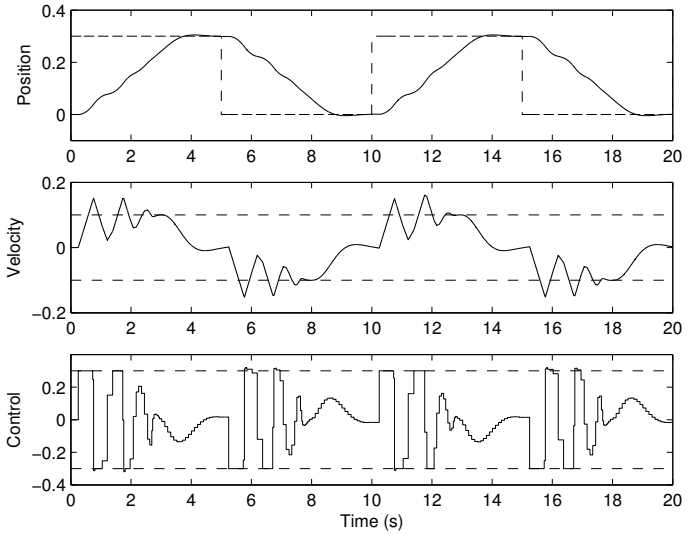


**Figure 3** Control performance when the optimization algorithm is allowed to finish in every sample. The bad performance is a result of considerable delay and jitter induced by the large variations in execution time. During the transients the long execution times cause the control task to miss its next invocation, inducing sampling jitter. The dashed lines in the velocity and control signal plots show the constraints used in the MPC formulation.

This implementation will allow for task overruns without aborting the ongoing computations. The control signal is actuated as soon as the task has completed.

In the dynamic scheduling scheme, the MPC task is divided into a mandatory and an optional part as described in Section 4. The mandatory part is scheduled with a distinct high priority, whereas the priority of the optional part is changed dynamically depending on the current value of the cost index in comparison to the other running MPC tasks.

### 5.2 Simulation of One MPC Controller

The first simulations consider the case of a single MPC task implemented according to the standard task model described in the previous section. Figure 3 shows the result of a simulation where the optimization is allowed to finish in each sample. Delay and jitter induced by the large variations in execution time compromise the optimal control performance. The constraints are shown by the dashed lines in the velocity and control signal plots. As seen in the plots the constraints are violated at some points. This is due to the computational delay, which is not accounted for in the MPC formulation.

Figure 4 shows a simulation where the termination criterion from Section 3 is exploited. The cost index (7) is evaluated after each iteration, and if it has increased since the last iteration, the optimization is aborted and the current control signal is actuated. As can be seen from the simulations, the control performance has increased significantly.

Figure 5 shows a comparison of the number of iterations needed for full optimization (top) and the number of iterations after which the optimization was aborted due to an increasing value of $J_d$ (bottom). The execution time of each iteration in the simulation was 10 ms. Average values for computation times

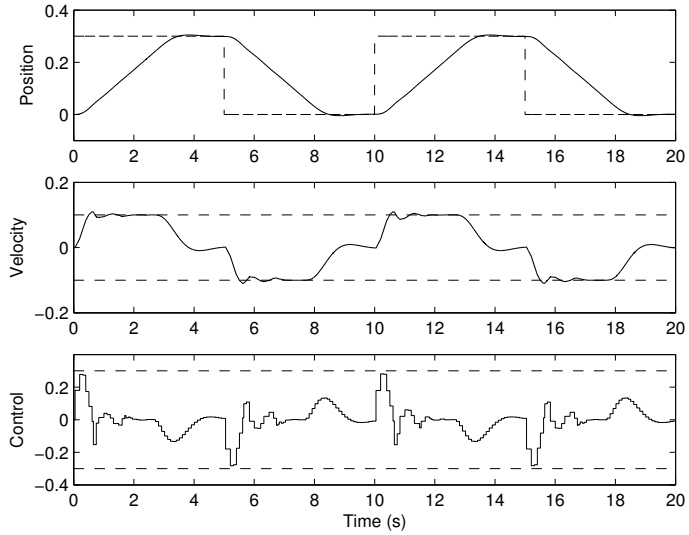


**Figure 4** Control performance obtained using the proposed sub-optimal approach where the QP-optimization may be aborted according to the termination criterion described in Section 3. The performance is increased substantially compared to Figure 3.

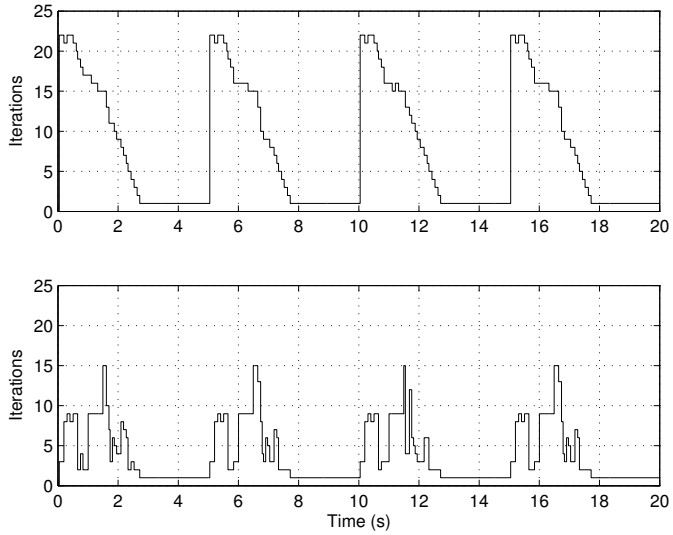


**Figure 5** Number of iterations for the QP-solver. The top plot shows the number of iterations to find the optimum. The bottom plot shows the number of iterations after which the optimization is terminated and the sub-optimal control is actuated.

and the number of iterations in the QP optimization algorithm in each sample is summarized in Table 1. The number of necessary iterations denotes the number of QP-iterations needed to fulfill the stability condition. It can be seen that the total execution time of the MPC task is reduced by 35 percent by using the proposed termination criterion. The execution time for the mandatory part of the algorithm is roughly constant for both approaches. In the full optimization case, the execution time will exceed the 100 ms sampling period during the transients, causing the control task to miss deadlines and experience sampling jitter.

**Table 1** Average timing values per sample for a simulation.

| Optimization | Full | Sub-optimal |
|---|---|---|
| Total time [s] | 0.1055 | 0.0692 |
| Mandatory time [s] | 0.0302 | 0.0297 |
| Number of iterations | 8.87 | 5.66 |
| Number of necessary iterations | 1.70 | 1.89 |

**Table 2** Performance loss comparison in the single MPC case.

| Strategy | Loss |
|---|---|
| Ideal case | 1.0 |
| Full optimization | 1.35 |
| Sub-optimal | 1.09 |

To quantify the simulation results, the *performance loss*

$$J = \int_0^{T_{sim}} \left( \|z(t) - r(t)\|_Q^2 + \|\Delta u(t)\|_R^2 \right) dt \tag{9}$$

was recorded in both cases. The weighting matrices, $Q$ and $R$, were the same as used in the MPC formulation. The performance loss was scaled with the loss for an ideal simulation. The ideal case was obtained by simulating full optimization and zero execution time in each sample. The results are given in Table 2.

### 5.3 Dynamic Scheduling of Two MPC Tasks

In the following simulations the dynamic scheduling strategy proposed in Section 4 will be compared to ordinary fixed-priority scheduling. Two MPC controllers are implemented and executed by two different tasks running concurrently on the same CPU controlling two different double-integrator processes. Both MPC controllers are designed with the same prediction and control horizons, sampling periods, and weighting matrices in the MPC formulation.

Both controllers were given square-wave reference trajectories, but with different amplitudes and periods. The reference trajectory for MPC1 had an amplitude of 0.3 and a period of 10 s. The corresponding values for MPC2 were 0.4 and 12 s. The different reference trajectories will cause the relative computational demands of the MPC tasks to vary over time. Therefore, it is not obvious which controller task to give the highest priority. Rather, this should be decided on-line based on the current state of the controlled process.

The simulation results are shown in Figures 6-8. The first two simulations show the fixed-priority cases. MPC1 is given the highest priority in the first simulation, and MPC2 is given the highest priority in the second simulation. It is seen that we get different control performance, depending on how we choose the priorities. By giving MPC2 the highest priority, the performance in this particular simulation scenario is considerably better than if the priorities are reversed.

The performance using dynamic scheduling based on the cost index (7) is shown in Figure 8, and the performance is improved significantly. Figure 9 shows a close-up of the computer schedule during one sample. After both tasks have

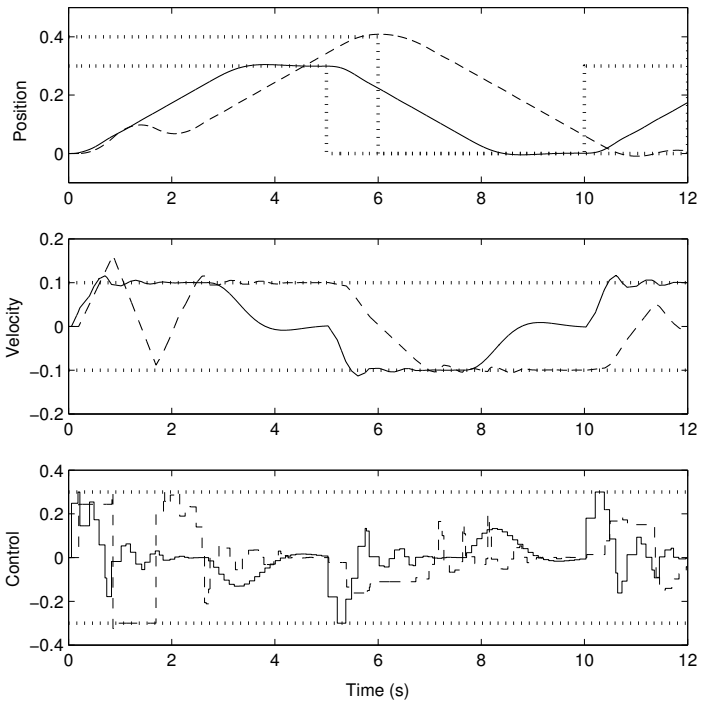


**Figure 6** Control performance using fixed-priority scheduling where MPC1 (solid) is given the highest priority. MPC2 (dashed) is constantly preempted by the higher priority task, consequently degrading its performance.

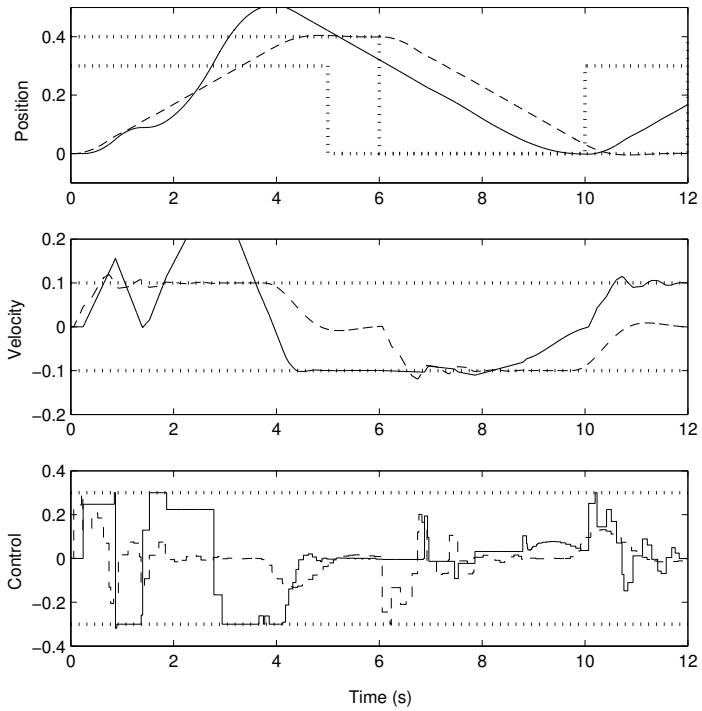


**Figure 7** Control performance using fixed-priority scheduling where MPC2 (dashed) is given the highest priority. Comparing with Figure 6 it can be seen that the performance is worse using this priority assignment.

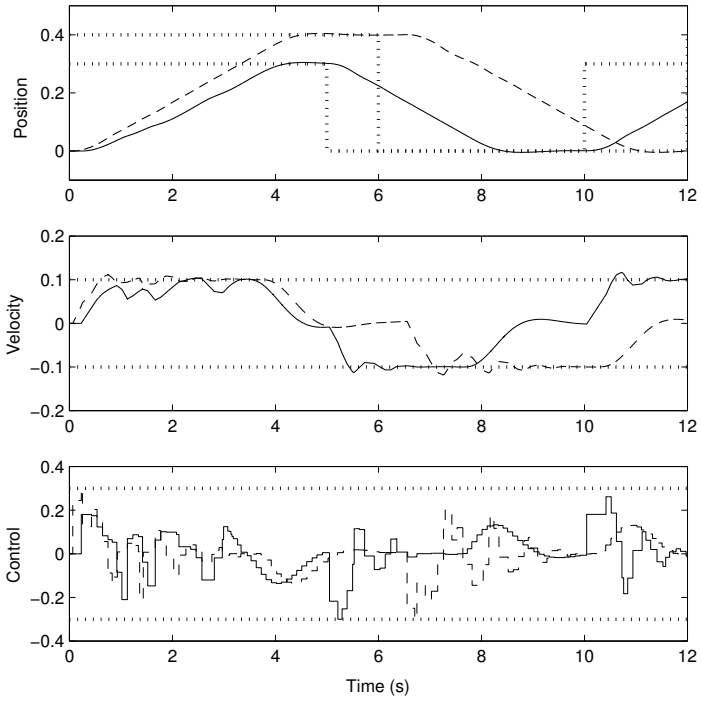


**Figure 8** Control performance using the dynamic scheduling approach. Scheduling based on cost functions makes sure that the most urgent task gets access to the processor, thus increasing the overall performance.

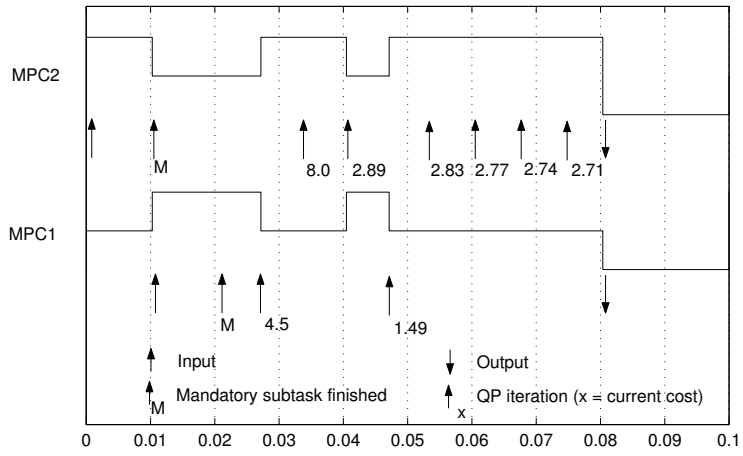


**Figure 9** Computer schedule in a sample using the dynamic scheduling approach (high = running, medium = preempted, low = idle). The figure shows the completion of the mandatory part, as well as the value of the cost index after each QP-iteration.

completed the mandatory parts of their algorithms, the execution trace (the dynamic priority assignments) is determined based on the values of the cost functions of the individual tasks. These values after each iteration are shown in the figure. The termination criterion aborts both tasks at time 0.08.

The scaled performance loss (9) for the individual control loops were added up to obtain a total loss for each scheduling strategy. The results are summarized in Table 3. It can be seen that the improvement using dynamic scheduling is

**Table 3** Performance loss for the different scheduling strategies.

| Strategy | Loss |
|---|---|
| Ideal case | 2.0 |
| Fixed priority / MPC1 highest priority | 2.47 |
| Fixed priority / MPC2 highest priority | 2.79 |
| Dynamic cost-based scheduling | 2.43 |

less significant in the case where MPC1 is given the highest priority. This is, however, due to the reference trajectories applied in this particular simulation. Also, in the fixed-priority case, the constraint on the state $x_2$ is significantly violated, which is not accounted for in the calculation of the cost index.

Using the proposed dynamic scheduling strategy we arbitrate the computing resources according to the current situation for the controlled processes, and the varying computational demands caused by reference changes and other external signals are taken into account at run-time. It should be noted that the control performance obtained using the dynamic cost-based scheduling would have been the same if the reference trajectories for the two controllers had been switched. As seen this is not the case using ordinary fixed-priority scheduling.

# 6. Conclusions

In this report we have shown how a novel termination criterion can be employed to improve the performance of sub-optimal, stabilizing MPC. A delay-dependent cost index has been presented that quantifies the trade-off between improved control signal quality resulting from successive iterations in the optimization algorithm and potential control performance degradation due to computational delay. The criterion provides guidance for when to terminate the optimization algorithm, while preserving the stability properties of the MPC algorithm.

It has also been shown how the cost index can be used in the context of dynamic real-time scheduling. The cost index has been used to provide the scheduling algorithm with information to be used for deciding which of two MPC controllers that should be allocated execution time. Using the index for scheduling, it has been shown how the overall control performance may be significantly improved compared to traditional fixed-priority scheduling.

# 7. References

Årzén, K.-E., A. Cervin, J. Eker, and L. Sha (2000): "An introduction to control and scheduling co-design." In *Proceedings of the 39th IEEE Conference on Decision and Control*. Sydney, Australia.

Åström, K. J. and B. Wittenmark (1997): *Computer-Controlled Systems*. Prentice Hall.

Bartlett, R. A., A. Wächter, and L. T. Biegler (2000): "Active set vs. interior point strategies for model predictive control." In *Proceedings of the American Control Conference*. Chicago, Illinois.

Bemporad, A., L. Chisci, and E. Mosca (1994): "On the stabilizing property of SIORHC." *Automatica*, **30:12**, pp. 2013–2015.

Bemporad, A., M. Morari, V. Dua, and E. N. Pistikopoulos (2002): "The explicit linear quadratic regulator for constrained systems." *Automatica*, **38:1**, pp. 3–20.

Buttazzo, G. C. (1997): *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers.

Cannon, M., B. Kouvaritakis, and J. A. Rossiter (2001): "Efficient active set optimization in triple mode MPC." *IEEE Transactions on Automatic Control*, **46:8**, pp. 1307–1312.

Cervin, A., J. Eker, B. Bernhardsson, and K.-E. Årzén (2002): "Feedback-feedforward scheduling of control tasks." *Real-Time Systems*, **23**, pp. 25–53.

Dunbar, W. B. and R. M. Murray (2002): "Model predictive control of coordinated multi-vehicle formations." In *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Dunbar, W. B., M. B. William, R. Franz, and R. M. Murray (2002): "Model predictive control of a thrust-vectored flight control experiment." In *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

Fletcher, R. (1991): *Practical methods of optimization 2nd ed.* John Wiley & Sons Ltd.

Garcia, C. E., D. M. Prett, and M. Morari (1989): "Model predictive control: Theory and practice – a survey." *Automatica*, **25:3**, pp. 335–348.

Henriksson, D., A. Cervin, J. Åkesson, and K.-E. Årzén (2002a): "On dynamic real-time scheduling of model predictive controllers." In *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Henriksson, D., A. Cervin, and K.-E. Årzén (2002b): "TrueTime: Simulation of control loops under shared computer resources." In *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

Kouvaritakis, B., M. Cannon, and J. Rossiter (2002): "Who needs QP for linear MPC anyway?" *Automatica*, **38:5**, pp. 879–884.

Liu, J., K.-J. Lin, W.-K. Shih, A. Yu, J.-Y. Chung, and W. Zhao (1991): "Algorithms for scheduling imprecise computations." *IEEE Trans on Computers*.

Liu, J., W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung (1994): "Imprecise computations." *Proceedings of the IEEE*, **82:1**, pp. 83–94.

Liu, J. W. S. (2000): *Real-Time Systems*. Prentice-Hall.

Maciejowski, J. M. (2002): *Predictive Control with Constraints*. Prentice-Hall.

Mayne, D. Q., J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert (2000): "Constrained model predictive control: Stability and optimality." *Automatica*, **36:6**, pp. 789–814.

Qin, S. J. and T. A. Badgwell (2003): "A survey of industrial model predictive control technology." *Control Engineering Practice*, **11:7**, pp. 733–764.

Richalet, J. (1993): "Industrial application of model based predictive control." *Automatica*, **29**, pp. 1251–1274.

Scokaert, P. O. M., D. Q. Mayne, and J. B. Rawlings (1999): "Suboptimal model predictive control (feasibility implies stability)." *IEEE Transactions on Automatic Control*, **44:3**, pp. 648–654.

Wright, S. J. (1997): *Primal-Dual Interior-Point Methods*. SIAM.