MPCtools 1.0 -- Reference Manual

Åkesson, Johan

2006

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

ISSN 0280–5316
ISRN LUTFD2/TFRT--7613--SE

MPCtools 1.0 — Reference Manual

Johan Åkesson

Department of Automatic Control
Lund Institute of Technology
January 2006

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

Document name
INTERNAL REPORT

Date of issue
January 2006

Document Number
ISRN LUTFD2/TFRT--7613--SE

Author(s)
Johan Åkesson

Supervisor

Sponsoring organization

Title and subtitle
MPCtools 1.0 – Reference Manual

Abstract
The maual describes MPCtools, a set a Matlab functions for simulation of MPC controllers. MPCtools supports linear state space models for prediction, linear inequality constraints on controls and states and quadratic cost functions. MPCtools also supports Simulink, enabling evaluation of MPC controllers applied to non-linear plants.

Keywords
Model Predictive Control (MPC), Matlab Toolbox, Simulink, Quadruple Tank

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
40

Recipient's notes

Security classification

Contents

1. Introduction 5

2. Getting Started 5

3. Model Predictive Control – Background 6

4. Linear Model Predictive Control 7

4.1 Receding horizon control 7

4.2 Model assumptions 8

4.3 An optimal control problem 8

4.4 Constraints 11

4.5 State estimation 12

4.6 Error-free tracking 12

4.7 Blocking factors 13

4.8 Linear properties of the MPC controller 14

5. Quadratic Programming Algorithms 15

6. MPCtools — Overview 16

6.1 A quadratic programming solver 17

6.2 MPCtools 17

7. Case Studies 20

7.1 The quadruple tank 20

7.2 The helicopter process 23

8. MPCTools Command Reference 26

MPCinit 27

MPCOptimizeSol 30

MPCSim 31

MPCController 32

MPCfrsp 33

qp_as 34

qp_ip 35

getfeasible 36

9. References 37

1. Introduction

This report describes a set of Matlab functions, referred to as MPCtools. MPCtools implements an MPC controller for use with Matlab/Simulink, with the following key features:

- Support for linear state space models for prediction
- Quadratic cost function
- Linear inequality constraints on states and controls
- Observer support for state and disturbance estimation
- Integral action by means of disturbance estimation
- Two different QP solvers for solving the optimization problem

The main purpose of MPCtools is to enable users to simulate and analyse control systems which includes MPC controllers, without having to get into the implementational details. In the development of MPCtools, ease of use has been considered more important than adding additional features.

MPCtools is primarily intended for research and teaching but is free to use for all purposes. The author accepts no responsibility for the consequences of using MPCtools.

2. Getting Started

MPCtools requires no installation or compilation, but it is convenient to add the directory containing the MPCtools functions to the Matlab path. The current distribution contains two application examples, demonstrating the key features of MPCtools. Both examples are described in detail in this report, and the corresponding scripts for setting up the problem and producing plots showing simulation results are included in the MPCtools distribution.

MPCtools are intended for use with Matlab R14, but should work also with Matlab R13. The tools require Control System Toolbox and, if the Simulink extension is to be used, also Simulink. The quadratic programming solver `quadprog` may be used to solve the MPC optimization problem, but this feature requires Optimization Toolbox. `quadprog` is not, however, necessary to use MPCtools.

The following steps show how to set up the path and how to run the example scripts.

1 Start Matlab

2 To add the directory containing MPCtools to the Matlab path, enter

```
>> addpath('/usr/matlab/MPCtools-1.0')
```

assuming that MPCtools reside in the directory `/usr/matlab/MPCtools-1.0`

3 Enter the quadruple tank example directory by entering

```
>> cd /usr/matlab/MPCtools-1.0/QuadTank
```

4 Run the script `Simulate_QuadTank_MPC` by entering

```
>> Simulate_QuadTank_MPC
```

to simulate a linear quadruple tank model controlled by an MPC controller.

5 Run the script `Simulate_QT_Simulink_MPC` by entering

```
>> Simulate_QT_Simulink_MPC
```

to simulate a nonlinear quadruple tank model controlled by an MPC controller. The script utilizes the Simulink model `QuadTank_MPC.mdl` for simulation.

6 Enter the helicopter example directory by entering

```
>> cd /usr/matlab/MPCtools-1.0/HelicopterII
```

7 Run the script `Simulate_Helicopter_MPC` by entering

```
>> Simulate_Helicopter_MPC
```

to simulate a linear nonlinear helicopter model controlled by an MPC controller.

3. Model Predictive Control – Background

The key feature that distinguishes MPC from most other control strategies is the receding horizon principle. An MPC controller solves, at each sampling instant, a finite horizon optimal control problem. Only the first value of the resulting optimal control variable solution is then applied to the plant, and the rest of the solution is discarded. The same procedure is then repeated at each sampling instant, and the prediction horizon is shifted forward one step. Thereby the name receding horizon control. This strategy includes solving *on-line* an optimal control problem, which enables the controller to deal *explicitly* with MIMO plants and constraints. On the downside are the computational requirements. Solving the optimization problem may introduce computational delay, which, if not considered, may degrade control performance. Also, MPC is a model based control strategy, and a model of the process to be controlled is a necessary requirement for MPC.

Historically, there has been two major selling points for MPC; it works well for MIMO plants, and it takes constraints into account *explicitly*. Both these issues arise frequently in many practical applications, and must be dealt with in order for a control design to be successful. MPC has been particularly successful in the area of process control, which is also the field from where MPC originates. Traditionally, MPC has been mainly applied to plants with slow dynamics, where the computational delay is small compared to typical sampling intervals. However, recent reports of MPC applications include plants with fast dynamics, ranging from air plane control to engine control. For a review of industrial use of MPC, including a historical review of the evolution of MPC, see [Qin and Badgwell, 2003]

During the last decade, there has been significant research efforts to sort out the theoretical issues of MPC. Notably, the problem of formulating a stabilizing MPC scheme has received much attention. As a result, several techniques

to ensure stability have been presented, see [Mayne *et al.*, 2000] for a review. The theory for MPC based on linear systems is well developed, and strong results ensuring robust stability exists, see [Maciejowski, 2002] for an overview. Also, the optimization problem resulting from linear MPC is a Linear Inequality Constrained Quadratic Programming (LICQP) problem, which is a convex optimization problem, and efficient solution algorithms exist. In particular, existence of a unique global minimum is guaranteed. For non-linear system, there exist MPC formulations that guarantee stability under mild conditions. However, the resulting optimization problem is, in general, non-convex and usually no guarantee of finding a global minimum exits. Non-linear MPC remains a very active field of research and recent results have shown that optimality is not a necessary condition for stability, see [Scokaert *et al.*, 1999]. These promising results show that many non-linear MPC schemes may be stabilizing although finding the global minimum is difficult.

In this report, Matlab tools for a standard formulation of linear MPC will be presented. State estimation, error-free tracking and stability are important issues that will be covered in the following. The aim of developing the MPC tools have been to simulate, in detail, the behavior of an MPC controller. A detailed analysis of controller behavior also requires understanding of the algorithm used to solve the LICQP problem. Accordingly, the tools include implementations of common LICQP algorithms for this purpose.

4. Linear Model Predictive Control

In this section, an MPC formulation based on linear discrete-time state space models will be described. The presentation is based on [Maciejowski, 2002].

4.1 Receding horizon control

The MPC scheme makes use of the receding horizon principle, illustrated in Figure 1. At each sample, a finite horizon optimal control problem is solved over a fixed interval of time, the prediction horizon. We assume that we would like the controlled variables, $z(k)$, to follow some set point trajectory, $r(k)$. The optimal control problem is formulated using a cost function penalizing deviations of the controlled variables as well as variations in the control signal. A common choice is to use a quadratic cost function, which in combination with a linear system model yields a finite horizon LQ problem. Figure 1 shows the predicted optimal trajectories $\hat{z}(k+i|k)$ and $\hat{u}(k+i|k)$ starting at time k.

The predictions necessary to solve the optimization problem is obtained using a model of the controlled system. The prediction of the controlled variable z is performed over an interval with length H_p samples. The control signal is assumed to be fixed after H_u samples. H_p and H_u are referred to as the prediction horizon and the control horizon respectively. The distinction between the prediction and control horizons is useful since the number of decision variables in the optimization problem increases with H_u, but is independent of H_p. Normally, $H_u < H_p$ in order to reduce the complexity of the optimization problem.

When the solution of the optimal control problem has been obtained, the value of the first first control variable in the optimal trajectory, $u(k|k)$, is applied to the process. The rest of the predicted control variable trajectory is discarded, and at the next sampling interval the entire procedure is repeated.

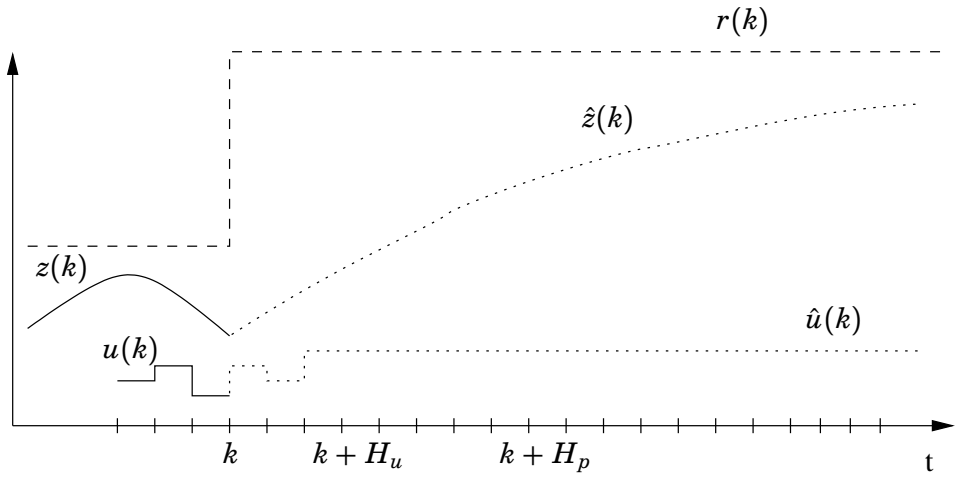

Figure 1 The idea of MPC. Here $r(k)$ is the set point trajectory, $z(k)$ represents the controlled output and $u(k)$ the control signal.

4.2 Model assumptions

We assume that a model on the form

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= C_y x(k) \\ z(k) &= C_z x(k) + D_z u(k) \\ z_c(k) &= C_c x(k) + D_c u(k) \end{aligned} \tag{1}$$

is available. Here $y(k) \in R^{p_y}$ is the measured output, $z(k) \in R^{p_z}$ the controlled output and $u(k) \in R^m$ the input vector. The state vector is $x(k) \in R^n$. The MPC controller should also respect constraints on control variables as well as the constrained outputs, $z_c(k) \in R^{p_c}$

$$\begin{aligned} \Delta u_{min} \leq \Delta u(k) \leq \Delta u_{max} \\ u_{min} \leq u(k) \leq u_{max} \\ z_{min} \leq z_c(k) \leq z_{max} \end{aligned} \tag{2}$$

where $\Delta u(k) = u(k) - u(k-1)$ are the control increments.

The distinction between controlled and constrained variables is natural, since only the controlled variables have specified reference values. This distinction is not made in [Maciejowski, 2002], but is quite useful. For example, there may be plant variables that must respect constraints, without having corresponding reference values. In some cases the constrained variables may not be included in the set of measured variables. In this case, an observer can be used to obtain estimates of such variables. The constraints are then enforced for the *estimated* outputs, which may not be equal to the true constrained outputs. The same argument applies to the controlled outputs, which will be discussed further in the section dealing with error free tracking.

4.3 An optimal control problem

We will now formulate the optimal control problem that is the core element of

the MPC algorithm. Consider the following quadratic cost function:

$$J(k) = \sum_{i=H_w}^{H_p+H_w-1} \|\hat{z}(k+i|k) - r(k+i|k)\|_Q^2 + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i|k)\|_R^2 \qquad (3)$$

where $\hat{z}(k+i|k)$ are the predicted controlled outputs at time k and $\Delta\hat{u}(k+i|k)$ are the predicted control increments. The matrices $Q \geq 0$ and $R > 0$ are weighting matrices, which are assumed to be constant over the prediction horizon. The length of the prediction horizon is H_p, and the first sample to be included in the horizon is H_w. H_w may be used to shift the control horizon, but in the following presentation we will assume that $H_w = 0$. The control horizon is given by H_u. In the cost function (3) $\Delta u(k)$ is penalized rather than $u(k)$, which is common in LQ control. The reason for this is that for a non-zero set point, $r(k)$, the corresponding steady state control signal $u(k)$ is usually also non-zero. By avoid penalizing $u(k)$, this conflict is avoided. A different method that has been used is to introduce a set point also for the control variable, $r_u(k)$, and to penalize deviations of $u(k)$ from u_r. This approach may be implemented in the above formulation by choosing $C_z = 0$ and $D_z = I$, and thereby let $u(k)$ be part of the controlled variables.

The cost function (3) may be rewritten as

$$J(k) = \|\mathcal{Z}(k) - \mathcal{T}(k)\|_{\mathcal{Q}}^2 + \|\Delta\mathcal{U}\|_{\mathcal{R}}^2$$

where

$$\mathcal{Z}(k) = \begin{bmatrix} \hat{z}(k|k) \\ \vdots \\ \hat{z}(k+H_p-1|k) \end{bmatrix} \quad \mathcal{T}(k) = \begin{bmatrix} r(k|k) \\ \vdots \\ r(k+H_p-1|k) \end{bmatrix}$$

$$\Delta\mathcal{U}(k) = \begin{bmatrix} \Delta u(k|k) \\ \vdots \\ \Delta u(k+H_u-1|k) \end{bmatrix} \quad \mathcal{Q} = \begin{bmatrix} Q & 0 & \dots & 0 \\ 0 & Q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Q \end{bmatrix}$$

$$\mathcal{R} = \begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R \end{bmatrix}$$

By deriving the prediction expressions, we can write

$$\mathcal{Z}(k) = \Psi x(k) + \Gamma u(k-1) + \Theta\Delta\mathcal{U}(k) \qquad (4)$$

where

$$\Psi = \begin{bmatrix} C_z \\ C_z A \\ C_z A^2 \\ \vdots \\ C_z A^{H_p-1} \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} D_z \\ C_z B + D_z \\ C_z AB + C_z B + D_z \\ \vdots \\ C_z \sum_{i=0}^{H_p-2} A^i B + D_z \end{bmatrix}$$

$$\Theta = \begin{bmatrix} D_z & 0 & \cdots & 0 \\ C_z B + D_z & D_z & & \\ C_z AB + C_z B + D_z & & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ C_z \sum_{i=0}^{H_u-2} A^i B + D_z & & \cdots & D_z \\ \vdots & & \ddots & \vdots \\ C_z \sum_{i=0}^{H_p-2} A^i B + D_z & & \cdots & C_z \sum_{i=0}^{H_p-Hu-1} A^i B + D_z \end{bmatrix}$$

Also, let

$$\mathcal{E}(k) = \mathcal{T}(k) - \Psi x(k) - \Gamma u(k-1).$$

This quantity could be interpreted as the free response of the system, if all the decision variables at $t = k$, $\Delta\mathcal{U}(k)$, were set to zero. Inserting the prediction expressions into the cost function (3) we obtain

$$J(k) = \Delta\mathcal{U}^T \mathcal{H} \Delta\mathcal{U} - \Delta\mathcal{U}^T \mathcal{G} + \mathcal{E}^T Q \mathcal{E} \tag{5}$$

where

$$\mathcal{G} = 2\Theta^T Q \mathcal{E}(k)$$
$$\mathcal{H} = \Theta^T Q \Theta + \mathcal{R}$$

The problem of minimizing the the cost function (5) is a quadratic programming (QP) problem. If $\mathcal{H}$ is positive definite, the problem is convex, and the solution may be written on closed form. Positive definiteness of $\mathcal{H}$ follows from the assumption that $Q \geq 0$ and $\mathcal{R} > 0$. The solution is given by

$$\Delta\mathcal{U} = \frac{1}{2}\mathcal{H}^{-1}\mathcal{G}.$$

Notice that the matrix $\mathcal{H}^{-1}$ does not depend on k, and may be pre-calculated. In fact, the controller is linear, and may be calculated off-line. This will be discussed in detail in Section 4.8.

4.4 Constraints

Let us now introduce constraints on the constrained and control variables, z_c and u. In general, linear constraints may be expressed as:

$$W\Delta\mathcal{U}(k) \leq w \tag{6}$$
$$F\mathcal{U}(k) \leq f \tag{7}$$
$$G\mathcal{Z}_c(k) \leq g \tag{8}$$
$$\tag{9}$$

This formulation allows for very general constraints, but in the following, only the constraints specified by (2) will be considered. Using (2), we obtain

$$W = F = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & & & \\ 0 & 1 & & \vdots \\ & -1 & & \\ \vdots & & \ddots & 0 \\ & & & 1 \\ 0 & \cdots & 0 & -1 \end{bmatrix} \quad G = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & & & \\ 0 & 1 & & \vdots \\ & -1 & & \\ \vdots & & \ddots & 0 \\ & & & 1 \\ 0 & \cdots & 0 & -1 \end{bmatrix}$$

$$w = \begin{bmatrix} \Delta u_{max} \\ -\Delta u_{min} \\ \vdots \\ \Delta u_{max} \\ -\Delta u_{min} \end{bmatrix} \quad f = \begin{bmatrix} u_{max} \\ -u_{min} \\ \vdots \\ u_{max} \\ -u_{min} \end{bmatrix} \quad g = \begin{bmatrix} z_{max} \\ -z_{min} \\ \vdots \\ z_{max} \\ -z_{min} \end{bmatrix}$$

Notice that W and F may not be of the same size as G, though the structure is the same. Since $\mathcal{U}(k)$ and $\mathcal{Z}_c(k)$ are not explicitly included in the optimization problem, we rewrite the above constraints in terms of $\Delta\mathcal{U}(k)$. This gives

$$\begin{bmatrix} \mathcal{F} \\ G\Theta_c \\ W \end{bmatrix} \Delta\mathcal{U} \leq \begin{bmatrix} -\mathcal{F}_1 u(k-1) + f \\ -G(\Psi_c x(k) + \Gamma_c u(k-1)) + g \\ w \end{bmatrix} \tag{10}$$

where

$$\mathcal{F} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & & & \\ 1 & 1 & & \vdots \\ -1 & -1 & & \\ \vdots & & \ddots & 0 \\ 1 & & & 1 \\ -1 & \cdots & & -1 \end{bmatrix} \quad \mathcal{F}_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ \vdots \\ 1 \\ -1 \end{bmatrix}$$

The definitions of Ψ_c, Θ_c and Γ_c are equivalent to those of Ψ, Θ and Γ. As we can see, the left side of the inequality is not dependent on k, and could be calculated off-line. The right side depends on the last control signal and the

present estimation of the state vector, and should thus be evaluated at each sample.

The optimization problem can now be rewritten using (3) and (10)

$$\begin{aligned} &\min J(k) = \Delta\mathcal{U}^T\mathcal{H}\Delta\mathcal{U} - \Delta\mathcal{U}^T\mathcal{G} + \mathcal{E}^T\mathcal{Q}\mathcal{E} \\ &\text{subject to } \Omega\Delta\mathcal{U} \leq \omega \end{aligned}$$

The problem is still recognized as a quadratic programming problem, but now with linear inequality constraints. The problem is convex, but due to the constraints, it is not possible to write the solution on closed form. Rather iterative algorithms have to be employed. This issue will be discussed further in Section 5

4.5 State estimation

The algorithm for obtaining the optimal control signal at each sample assumes that the present state vector is available. Since this is often not the case, state estimation is required. The celebrated separation principle, stating that the optimal control and optimal estimation problems solved independently, yields a globally optimal controller for linear systems, suggests an attractive approach. We let the solution of the optimization problem be based on an estimate of the state vector, $\hat{x}(k)$ instead of the true state vector $x(k)$. For this purpose, a Kalman filter,

$$\hat{x}(k+1) = A\hat{x}(k) + Bu(k) + K(y(k) - C_y\hat{x}(k)).$$

can be used. If the covariance matrices of the states, W, and the measurement noise, V, are assumed to be known, the gain matrix K may be obtained by solving an algebraic Riccati equation, see for example [Åström and Wittenmark, 1990].

Apart from estimating the state of the system, an estimator could be used to estimate disturbances, assuming that a disturbance model is available. For example, error-free tracking may be achieved by including a particular disturbance model in the observer.

4.6 Error-free tracking

In practical applications, there are always modeling errors and disturbances present. The MPC formulation described above contains no explicit mechanism to deal with these complications. In order for the controller to be useful in practice, these problems have to be considered. Commonly, the controller is designed so that it contains integral action, which ensures zero steady-state error. There are several methods for achieving integral action in a controller. For SISO systems, introduction of integral action is quite straight forward. A common approach is to introduce an integrator state in the state space model:

$$\begin{aligned} \begin{bmatrix} x(k+1) \\ x_i(k+1) \end{bmatrix} &= \begin{bmatrix} A & 0 \\ -C_z & I \end{bmatrix} x(k) + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ I \end{bmatrix} r(k) \\ y(k) &= \begin{bmatrix} C_y & 0 \end{bmatrix} \\ z(k) &= \begin{bmatrix} C_z & 0 \end{bmatrix}. \end{aligned}$$

A stabilizing feedback control law may then be calculated based on the extended model. The integrator state is implemented in the controller, and used for feedback together with the true or estimated states. From the definition of the extended system model, it is clear that in steady state, $z = r$. This approach does

not work so well for MPC controllers. In particular, it is not clear how the integral state should be introduced in the cost function in order for the integral action to work properly.

A different approach to achieve integral action is to use a disturbance observer. In summary, the extended system model

$$\begin{bmatrix} x(k+1) \\ v_a(k+1) \\ d(k+1) \end{bmatrix} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ v_a(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix} u(t)$$

$$z(k) = y_z(k) = \begin{bmatrix} C_z & 0 & 0 \end{bmatrix} \begin{bmatrix} x(k)^T & v_a(k)^T & d(k)^T \end{bmatrix}^T$$

$$y_a(k) = \begin{bmatrix} C_a & I & 0 \end{bmatrix} \begin{bmatrix} x(k)^T & v_a(k)^T & d(k)^T \end{bmatrix}^T$$

is used. It is assumed that the number of controlled outputs, z, equals the number of inputs, u. Additional outputs, if any, are denoted y_a. Also, the controlled variables are assumed to be included in the set of measured variables. A detailed treatment of this method is given in [Åkesson and Hagander, 2003].

4.7 Blocking factors

In some situations it may be advantageous to let the control signal be fixed over several consecutive predicted samples. In this way, the control horizon may be increased without increasing the complexity of the optimization problem. Also, ringing behavior of the control signal may be avoided. For example, suppose that the control horizon has to be increased in order to increase closed loop performance. If the control horizon is increased, the time to solve the optimization problem will also increase. If this is not acceptable, one approach might be to include only every other decision variable in the optimization problem, assuming that the the control signal is fixed over two consecutive sampling intervals. We denote the set of predicted sample indexes for which the control signal is allowed to vary by I_u.

In the MPC formulation given above this means that some $\Delta\hat{u}(k+i|k)$:s are set to zero for certain i:s. This means that the corresponding columns in the matrices Θ, W and F may be neglected.

In a similar way it is possible to generalize the prediction horizon. Instead of including all predicted values in the interval $[k...k+H_p-1]$, we introduce the set I_p consisting of all sample indexes for which the corresponding predicted output values are included in the cost function and for which the constraints are enforced. The last point is critical. It may be tempting to introduce a sparse set of predicted sampling instants in order to obtain a longer prediction horizon. However, since the inter-sample behavior is neglected, this may lead to the constraint in effect being violated at some points.

This generalization is easily introduced by neglecting the rows of the matrices Ψ, Γ, Θ and G corresponding to sample indexes not present in I_p.

Using the notation introduced above, the cost function may be rewritten as

$$J(k) = \sum_{i \in I_p} \|\hat{z}(k+i|k) - r(k+i|k)\|_Q^2 + \sum_{i \in I_u} \|\Delta\hat{u}(k+i|k)\|_R^2 . \qquad (11)$$

4.8 Linear properties of the MPC controller

The behavior of the MPC controller is intrinsically nonlinear, since constraints on state and control variables are taken into account. However, if no constraints are present in the problem formulation, the controller is linear. Also, the controller behaves linearly during operation when no constraints are active. In the first case, the control law, could (and should) be calculated off-line, whereas in the second case, the optimization procedure must be done each sample. There are however, methods for avoiding on-line solution of the optimization problem. Using the observation that the MPC control law is piecewise linear in the states, it is possible to calculate, off-line, all possible control laws. The on-line optimization problem is then transformed into a search problem, where the objective is to find the appropriate partition in the state space, identifying the corresponding control law. This approach is described in [Bemporad *et al.*, 2002].

We will now analyze the linear properties of the MPC controller. The analysis is valid for the case when no constraints are present or the controller operates so that no constraints are active. In this case, the minimizing solution of the quadratic programming problem is

$$\begin{aligned}\Delta\mathcal{U}(k) &= (\Theta^T Q\Theta + \mathcal{R})^{-1}\Theta^T Q\mathcal{E}(k) \\ &= (\Theta^T Q\Theta + \mathcal{R})^{-1}\Theta^T Q \begin{bmatrix} \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix} & -\Gamma & -\Psi \end{bmatrix} \begin{bmatrix} r(k) \\ u(k-1) \\ \hat{x}(k) \end{bmatrix} \\ &= \bar{K}_s \begin{bmatrix} r(k) \\ u(k-1) \\ \hat{x}(k) \end{bmatrix}\end{aligned}$$

Now, since only the first of the predicted control signals are applied we can write the control law as

$$\begin{aligned}\Delta u(k|k) = \Delta u(k) &= K_s \begin{bmatrix} r^T(k) & u^T(k-1) & \hat{x}^T(k) \end{bmatrix}^T \\ &= \begin{bmatrix} K_{sr} & K_{su} & K_{sx} \end{bmatrix} \begin{bmatrix} r^T(k) & u^T(k-1) & \hat{x}^T(k) \end{bmatrix}^T\end{aligned}$$

where K_s is given by the first m rows of $\bar{K}_s$. This control law is linear, and the constant gain matrix K_s may be calculated off-line. The block diagram of the controller may now be drawn as in Figure 2. In this figure, the transfer function (matrix) of the plant is given by $P(z)$, and $H_u(z)$ and $H_y(z)$ represents the observer. This block diagram is readily converted into a feedback system on standard form shown in Figure 3, with

$$\begin{aligned}P(z) &= C_y(zI - A)^{-1}B \\ F(z) &= K_{sr} \\ H(z) &= -K_{sx}H_y(z) \\ K(z) &= \frac{z}{z-1}\left[I - \frac{1}{z-1}K_{su} - \frac{z}{z-1}K_{sx}H_u(z)\right]^{-1} \\ H_y(z) &= (zI - A + KC_y)^{-1}K \\ H_u(z) &= (zI - A + KC_y)^{-1}B\end{aligned}$$

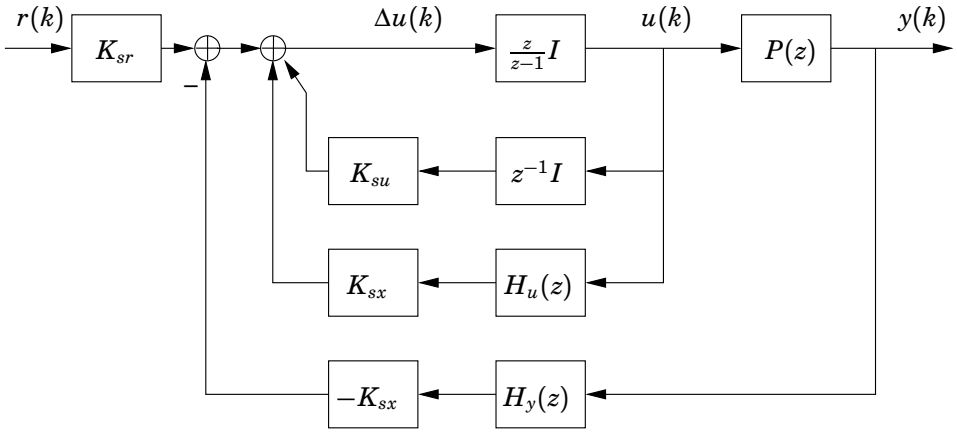

Figure 2 The block diagram for the MPC controller

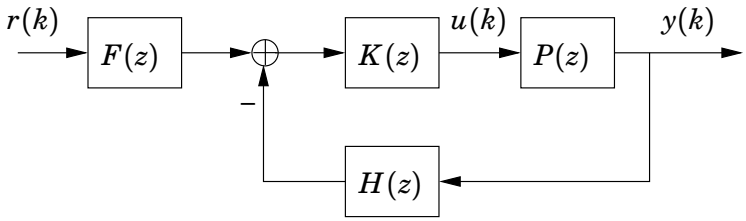

Figure 3 A standard feedback structure.

It is now straight forward to apply standard linear analysis methods. For example, the poles and zeros of the closed loop system may be calculated, as well as the sensitivity of the system.

5. Quadratic Programming Algorithms

An important element of the MPC algorithm described above is the algorithm for solving the LICQP problem. The problem at hand is

$$\begin{aligned} &\min J(k) = \Delta\mathcal{U}^T \mathcal{H} \Delta\mathcal{U} - \Delta\mathcal{U}^T \mathcal{G} + \mathcal{E}^T \mathcal{Q} \mathcal{E} \\ &\text{subject to } \Omega\Delta\mathcal{U} \leq \omega. \end{aligned}$$

This problem has several nice features. For example, the objective function is convex, since it is quadratic with positive definite Hessian. Also, the constraints are also convex. Given these conditions, it is a well known result that a local minimum, if it exists, is also a global minimum. (See for example Theorem 4.3.8 in [Bazaraa *et al.*, 1993].) When designing numerical algorithms, this property is of course very valuable, since we know in advance that if we find a minimum, it is indeed a global minimum.

There exist several algorithms for constrained optimization, see for example [Fletcher, 1987]. For quadratic programming problem the two most common approaches are primal-dual internal point methods and active set methods [Maciejowski, 2002].

The active set algorithm assumes an initial point in the decision variable space that fulfills the constraints. The active set is defined as the set of all active constraints at this point. A constraint is said to be active if a particular point in the search space is at the boundary of the feasible region defined by the constraint. In the case of linear constraints, these boundary surfaces are given by hyper-planes. In each iteration step, a quadratic programming problem with linear equality constraints (namely those in the active set) is solved. The solution of this problem may be written on closed form. Possibly this solution leads to the introduction of a new constraint into the active set. By calculating the Lagrange multipliers for the problem at each iteration, it is possible to conclude if a constraint may be relaxed, that is, removed from the active set. The algorithm terminates when the gradient of the associated Lagrange function is identically zero, and all Lagrange multipliers are positive.

One problem remains to deal with; the feasible initial point. In order to start the active set algorithm, we need a feasible solution, that is, we would like to find a solution that fulfills the constraints $\Omega\Delta\mathcal{U} \leq \omega$. Of course, such a solution is not likely to be unique. Several methods exist for obtaining the desired solution. For example, the problem may be cast as an LP problem, and solved by the simplex algorithm. Another and possibly more attractive alternative is given in [Fletcher, 1987]. This strategy employs an active set technique similar to the one for solving the main quadratic programming problem. However, in this case, the objective function is defined at each iteration as the sum of the violated constraint functions.

Primal-dual interior point methods on the other hand, explores the Karush-Kuhn-Tucker conditions explicitly. The name of this family of QP algorithms stems from the fact that the primal and the dual problems are solved simultaneously. It is important to note however, that the term *interior point* refers to the fact that the algorithm maintains a solution in the interior of the *dual* space. In fact, the primal solution may not be feasible during the optimization run, except at the optimal point. This constitutes an important difference between active set methods and primal-dual interior point methods. Specifically, in the former case it is possible to terminate the optimization algorithm prematurely and still obtain a feasible, but sub-optimal, solution whereas in the latter case, the algorithm may terminate only when the optimal solution is found.

Concerning performance, both methods have advantages and disadvantages when applied to MPC. Rather, the key to achieve good performance lies in exploring the special structure of the MPC QP problem. See [Bartlett *et al.*, 2000] for a comparison between interior point and active set methods.

6. MPCtools — Overview

In this section, Matlab tools implementing the algorithms described in the previous sections are presented. The main objective for implementing the MPC tools has been to enable detailed study of the behavior of an MPC controller. In this section, the functionality of the tools is described briefly, see Section 8 for a detailed description.

6.1 A quadratic programming solver

Obviously, a quadratic programming solver is essential for the implementation of the MPC controller. A solver of this type is available for example from the Matlab Optimization Toolbox; `quadprog`. However, the aim of the development of the Matlab tools has been to create a complete implementation of an MPC controller. Since this also includes an algorithm to solve the QP problem, a solver based on active sets as well as a primal-dual interior point QP solver have been implemented. In addition, an algorithm for finding an initial feasible solution for the active set solver has been implemented. The following Matlab functions implement the necessary algorithms:

- `getfeasible`
  This function obtains a feasible point given the constraints $Ax \leq b$. The algorithm is described in [Fletcher, 1987, p. 166].
- `qp_as`
  This is an active set solver, based on [Fletcher, 1987, p. 240]. The algorithm finds the solution of the optimization problem

$$\begin{aligned} &\min \frac{1}{2}x^T Hx + f^T x \\ &\text{s.t.} \\ &Ax \leq b \end{aligned}$$

  using an initial solution x_0 as starting point. The possibility of starting the algorithm from an initial solution is quite useful when implementing the MPC controller. Usually, the solution obtained at the previous sample is a good initial guess, and the number of iterations may be reduced by supplying this solution to the QP algorithm
- `qp_ip`
  This function solves the same quadratic problem problem as `qp_as`. The algorithm implements the Mehrotra predictor-corrector primal-dual interior point method based on [Wright, 1997].

The MPC controller described in the next section enables the user to explore either quadratic programming method, as well as the Optimization Toolbox function `quadprog`.

6.2 MPCtools

The Matlab functions presented in this section implement an MPC controller as described in Section 4. The tools enable the user to simulate a linear or nonlinear plant model, with an MPC controller engaged. Most of the features described in Section 4 are implemented, including constraint handling, observer support, blocking factors and error-free tracking. Also, a basic tool for analyzing the linear properties of the controller is supplied.

In summary, the same assumptions as in Section 4 are made for the Matlab implementation of the MPC controller. We assume that a linear discrete time model on the form (1), with linear inequality constraints (2) is available. Also, we assume that the design variables H_w, H_p, H_u and the blocking factors are specified, as well as the weighting matrices Q, R and, if applicable, W and V for the design of a Kalman filter.

An important goal in designing the MPC tools has been to enable the user to experiment with different configurations of the MPC controller. Consequently, the MPC tools support several modes of operation:

- `Mode 0`: State feedback.
- `Mode 1`: State feedback with explicit integrators.
- `Mode 2`: Observer-based output feedback.
- `Mode 3`: Observer-based output feedback with explicit integrators.
- `Mode 4`: Observer-based output feedback with a disturbance model that gives error free tracking.

Both state and output feedback is supported. Also, two different strategies for achieving error-free tracking are provided; explicit integrators and an observer based solution. Both methods are described in Section 4.

The MPC tools are implemented in five functions:

- `MPCInit`
  Typically, `MPCInit` is used to initialize the MPC controller. Most of the matrices needed to solve the optimization problem may be calculated off-line in advance, for example the Hessian of the QP problem. `MPCInit` implements pre-processing of matrices, which drastically improves the performance of the controller.
- `MPCSim`
  `MPCSim` simulates a linear plant model controlled by the MPC controller. Reference trajectories as well as input disturbance trajectories can be supplied. The function produces the state and control variable trajectories, as well as the predicted trajectories for the controlled and control variables. The latter feature enables the user to examine the predicted solutions obtained at each sample, and explore the effect of different choices of prediction horizons.
- `MPCfrsp`
  As noted in Section 4.8, the MPC controller behaves linearly if no constraints are active. It is then interesting to use standard tools for analysis of linear systems. The function `MPCfrsp` provides frequency response plots for relevant transfer functions corresponding to, e.g., the closed loop system and input and output sensitivity functions.
- `MPCOptimizeSol`
  `MPCOptimizeSol` is a low level function used by other functions rather than by the user. This function provides the control signal of the MPC controller given the current state measurement (or estimation) and reference value.
- `MPCController`
  Most real world plants are nonlinear. It is therefore of interest to investigate the performance of the linear MPC controller applied to a nonlinear plant model, if it is available. This functionality is provided by the simulation environment Simulink for Matlab. The MPC controller has been adapted to Simulink, and is implemented by a Matlab S-function. The S-function

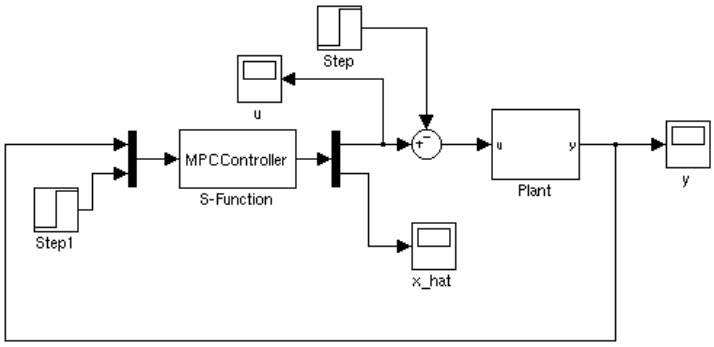

Figure 4 A Simulink model where the MPC controller is used to control a nonlinear plant.

is a standard block in Simulink, which may in turn be connected to arbitrary blocks. For an example, see Figure 4. The name of the S-function is `MPCController`, and is intended for use with the Simulink S-function block. It takes the argument `md`, which is a data structure created by the `MPCInit` function. The `MPCController` block takes as its inputs a vector signal consisting of the measured outputs of the plant and the reference value. Its output is a vector signal consisting of the control signal u, and the internal state estimation of the controller, $\hat{x}$. The latter may, apart from the estimated states of the plants, also include estimated disturbances or integrator states.

Using the MPC tools described above, it is possible to simulate, at a high level of detail, the behavior of an MPC controller. For example, the consequences of various assumptions regarding measurements, integral action schemes, choice of QP algorithm and prediction horizons are easily explored. Also, the use of Simulink significantly increases the applicability of the tools, enabling the user to simulate the behavior of the MPC controller when applied to nonlinear plants.

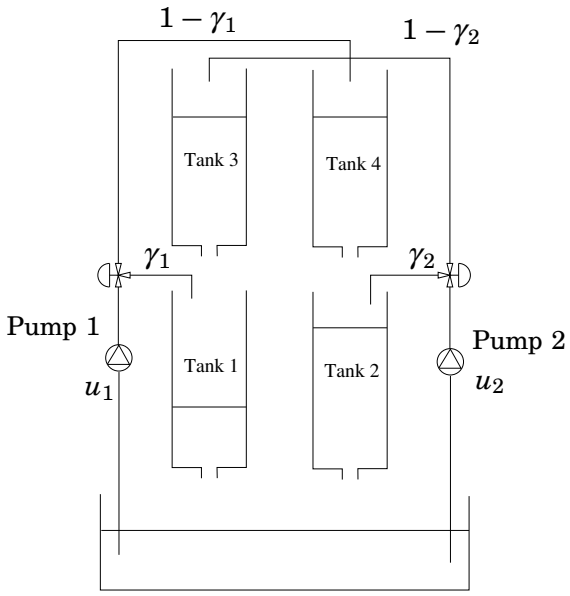

Figure 5 A schematic picture of the quadruple tank process

7. Case Studies

In this section, two cases studies are reported, illustrating the main functionality of the MPC tools.

7.1 The quadruple tank

The quadruple-tank laboratory process, see Figure 5, was originally presented in [Johansson, 1997]. The process consists of four tanks, organized in pairs (left and right), where water from the two upper tanks flows into the two lower tanks. A pump is used to pour water into the upper left tank and the lower right tank. A valve width fixed position is used to allocate pump capacity to the upper and lower tank respectively. A second pump is used to pour water into the upper right tank and lower left tank. The control variables are the pump voltages. Let the states of the system be defined by the water levels of the tanks (expressed in cm) x_1, x_2, x_3 and x_4 respectively. The maximum level of each tank is 20 cm. The dynamics of the system is given by

$$\begin{aligned}
\dot{x}_1 &= -\frac{a_1}{A_2}\sqrt{2gx_1} + \frac{a_3}{A_1}\sqrt{2gx_3} + \frac{\gamma_1 k_1}{A_1}u_1 \\
\dot{x}_2 &= -\frac{a_2}{A_2}\sqrt{2gx_2} + \frac{a_4}{A_2}\sqrt{2gx_4} + \frac{\gamma_2 k_2}{A_2}u_2 \\
\dot{x}_3 &= -\frac{a_3}{A_3}\sqrt{2gx_3} + \frac{(1-\gamma_2)k_2}{A_3}u_2 \\
\dot{x}_4 &= -\frac{a_4}{A_4}\sqrt{2gx_4} + \frac{(1-\gamma_1)k_1}{A_4}u_1
\end{aligned} \tag{12}$$

where the A_i:s and the a_i:s represent the cross section area of the tanks and the holes respectively. The parameters γ_i:s determine the position of the valves which control the flow rate to the upper and lower tanks respectively. The control signals are given by the the u_i:s. The objective is to control the levels of the two

Table 1 Parameter values of the Quadraple Tank

Parameters	Values	Unit
A_1, A_2	28	[cm^2]
A_3, A_4	32	[cm^2]
a_1, a_2	0.071	[cm^2]
a_3, a_4	0.057	[cm^2]
k_1, k_2	3.33, 3.35	[cm^3/Vs]
k_c	0.50	[V/cm]
g	981	[cm/s^2]

lower tanks, i.e. x_1 and x_2. Numerical values of the parameters are given in Table 1.

An interesting feature of this multivariate process is that the linearized system has an adjustable zero. By adjusting the parameters γ_1 and γ_2 it is possible to obtain a zero with negative real part, or a non minimum phase zero with positive real part. For the simulations, the valve positions were set to $\gamma_1 = 0.30$ and $\gamma_2 = 0.30$, yielding a non-minimum phase system.

The process is nonlinear, and in order to apply the MPC tools, a linearized model must be derived. Introducing $\Delta x = x - x^0$, $\Delta u = u - u^0$ and $\Delta y = y - y^0$, we obtain

$$\begin{aligned} \Delta \dot{x} &= \begin{bmatrix} -\frac{1}{T_1} & 0 & \frac{A_4}{A_1 T_3} & 0 \\ 0 & -\frac{1}{T_2} & 0 & \frac{A_4}{A_2 T_4} \\ 0 & 0 & -\frac{1}{T_3} & 0 \\ 0 & 0 & 0 & -\frac{1}{T_4} \end{bmatrix} \Delta x + \begin{bmatrix} \frac{\gamma_1 k_1}{A_1} & 0 \\ 0 & \frac{\gamma_2 k_2}{A_2} \\ 0 & \frac{(1-\gamma_2) k_2}{A_3} \\ \frac{(1-\gamma_1) k_1}{A_4} & 0 \end{bmatrix} \Delta u \\ \Delta y &= \begin{bmatrix} k_c & 0 & 0 & 0 \\ 0 & k_c & 0 & 0 \end{bmatrix} \Delta x \end{aligned} \tag{13}$$

where

$$T_i = \frac{A_i}{a_i} \sqrt{\frac{2 x_i^0}{g}}.$$

The stationary operating conditions, x^0 and u^0, are given in Table 2. The particular choice of valve positions $\gamma_1 = 0.25$ and $\gamma_2 = 0.35$ yields a non-minimum phase system. The measured outputs of the system are the levels of the lower tanks, represented by a voltage ranging from 0 to 10 V. The objective of the control system is to control, independently, the level of the lower tanks, while preventing overflow in any of the four tanks. The maximum level of the tanks is 20 cm, corresponding to 10 V. In the simulations, the tank level constraints were set to 19.8 cm to ensure some safety margin. Also, the operation of the pumps is limited to 0 to 10 V. We notice that the stationary level of the second tank is close to the maximum level. The combination of a non-minimum phase system and an operating point close to a constraint yields a quite challenging control problem, where two of the main benefits of the MPC controller, namely constraint handling and MIMO support, will be useful. The plant was discretized using the sampling interval $h = 3$ s, resulting in a discrete time model used for the MPC control design.

Table 2 Stationary values corresponding to $\gamma_1 = 0.25$ and $\gamma_2 = 0.35$.

Variables	Values	Unit
x_1^0, x_2^0	8.2444, 19.0163	[cm]
x_3^0, x_4^0	4.3146, 8.8065	[cm]
u_1^0, u_2^0	3, 3	[V]

Table 3 MPC controller parameters for the Quadruple Tank simulations

Parameter	Value
H_p	30
H_w	1
H_u	10
I_p	Only every other sample was included in the optimization problem.
I_u	Every other control increment was assumed to be constant.
Q	diag(4, 1)
R	diag(0.01, 0.01)
W	diag(1, 1, 1, 1)/diag(1, 1, 1, 1, 1, 1)
V	diag(0.01, 0.01)

Control of the linearized model In Table 3, the controller parameters used in the simulations are summarized. The choice of prediction and control horizons have been made considering the time constants of the system. Too short horizons may cause instability. However, the prediction and control horizons must not be chosen to large, since it would result in an unnecessarily complex QP problem to solve at each sample. In order to increase the horizons without increasing the number of optimization variables, the blocking factor 2 has been specified for both the control and prediction horizons.

Obviously, since all states are not measurable, an observer must be used. In the first simulation, controller mode 2 was used: a Kalman filter was designed, but no mechanism to ensure integral action was assumed. The result of the simulation is shown in Figure 6, represented by the dashed curves. The dotted curves represent the set points for the lower tanks. A step change in the set point of level 1 of size 6 cm is applied at $t = 60$ s, while the set point of level 2 is held constant. As we can see, the controller achieves the correct set points, and in particular, the level of tank 2 does not exceed 20 cm. At $t = 600$ s, a unit step disturbance is applied to the input channel 2. As expected, the MPC controller does not manage to achieve error-free tracking in the presence of load disturbances.

In a second design, control mode 4 was assumed. In this case, the disturbance observer described in [Åkesson and Hagander, 2003] was used to estimate the state vector and the disturbance states. The response of the system is identical to the previous case during the step response. However, the input disturbance is now rejected.

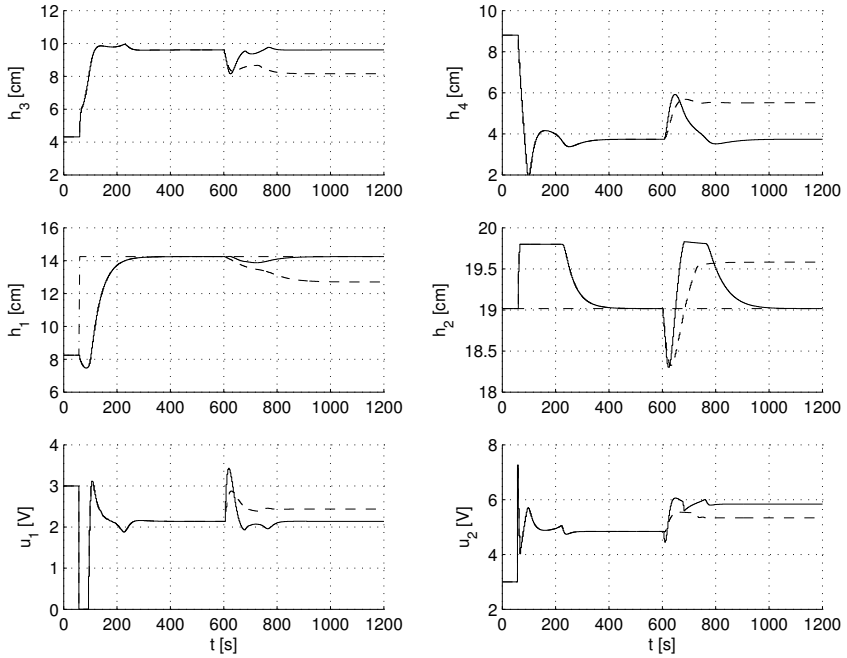

Figure 6 Simulations of the MPC controller applied to the linearized plant.

Control of the nonlinear model As noted above, the true Quadruple Tank system is nonlinear. A more realistic scenario would then be to apply the MPC controller designed above to the nonlinear plant model. The plant model was implemented in Simulink, and an S-function block representing the MPC controller was connected to the plant model block. The design parameters of the MPC controller were identical to the previous case, see Table 3. Also the same simulation scenario was assumed. The result of the simulation can be seen in Figure 7. The dashed curves represent the case when an observer without disturbance states is employed. As we can see, the controller fails to achieve error-free tracking even when no disturbances are present, which is due to the fact that there is a model-mismatch between the linear and the nonlinear plant. Obviously, the controller also fails to compensate fully for the load disturbance. If the disturbance observer is employed however, the controller gives zero steady-state error. This is shown by the solid curves. In both cases, however, the controller respects the constraints: no tank level exceed 20 cm.

7.2 The helicopter process

As an example of a process with fast dynamics, we consider the helicopter process shown in Figure 8. The helicopter consists of an arm mounted to a base, enabling the arm to rotate freely. The arm carries the helicopter body on one end and a counterweight on the other end. The helicopter body consists of a bar connected to the arm in such a way that it may rotate around the arm axis. To the bar is connected two propellers, which may be used to maneuver the plant. A schematic image of the process is shown in Figure 8, where the elevation angle, θ_e, measures the angle of the arm with respect to the horizontal plane, the travel angle, θ_r, measures the rotational position of the arm and the pitch angle, θ_p, measures

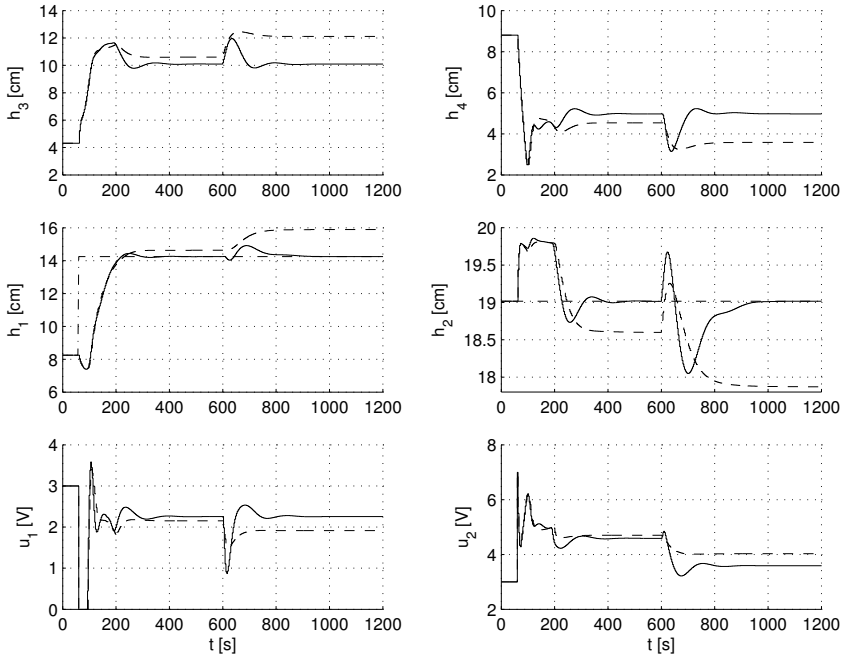

Figure 7 Simulations of the MPC controller applied to the nonlinear plant.

Table 4 Parameter values of the helicopter process

Parameter	Value	Unit	Description
J_e	0.91	[kgm^2]	Moment of inertia about elevation axis
l_a	0.66	[m]	Arm length from elevation axis to helicopter body
K_f	0.5		Motor Force Constant
F_g	0.5	[N]	Differential force due to gravity and counter weight
T_g	$l_a F_g$	[Nm]	Differential torque
J_p	0.0364	[kgm^2]	Moment of inertia about pitch axis
l_h	0.177	[m]	Distance from pitch axis to either motor
J_t	0.91	[kgm^2]	Moment of inertia about travel axis

the angle of the bar. A simple dynamical model of the system is given by

$$\begin{aligned} \ddot{\theta}_e &= (K_f l_a / J_e)(V_f + V_b) - T_g / J_e \\ \ddot{\theta}_r &= -(F_g l_a / J_t) \sin \theta_p \\ \ddot{\theta}_p &= (K_f l_h / J_p)(V_f - V_b) \end{aligned} \tag{14}$$

where the coefficients are given in Table 4. The input signals to the system are V_f and V_b which represent the voltages fed to the propeller motors. In addition, the elevation and pitch angles are constrained, so that

$$-0.5 \leq \theta_e \leq 0.6 \quad -1 \leq \theta_p \leq 1.$$

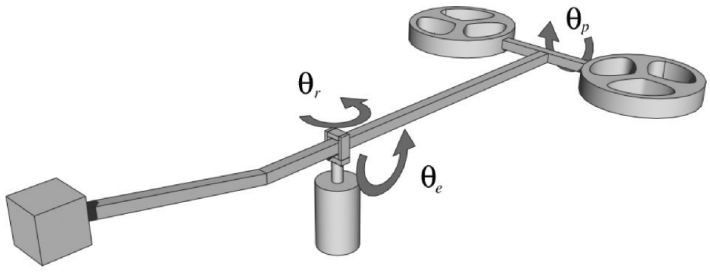

Figure 8 The helicopter process.

Table 5 MPC controller parameters for the Helicopter simulations

Parameter	Value
H_p	30
H_w	1
H_u	10
I_p	Only every other sample was included in the optimization problem.
I_u	Every other control increment was assumed to be constant.
Q	diag(1,1)
R	diag(0.1,0.1)

The process was linearized around the stationary point

$$\left(\theta_e^0,\ \theta_r^0,\ \theta_p^0,\ V_f^0,\ V_b^0,\right) = \left(0,\ 0,\ 0,\ T_g/(2K_f l_a),\ T_g/(2K_f l_a)\right)$$

and then discretized using the sampling interval $h = 0.2$ s. This gives a discrete-time state space model of the process to be used to design the MPC controller.

The task of the control system is to control, independently, the elevation angle and the rotation angle. The main constraint is that the pitch angle must not be too large, reflecting the fact that for large θ_p, the control authority in the θ_e direction is small. This is not accounted for in the linearized model, and the controller is likely to have degraded performance when θ_p is large.

The design parameters of the MPC controller are given in Table 5. Again, the choice of prediction and control horizons have been made considering the dominating time constants of the system. We also assume that the entire state vector is measurable, enabling the use of the state feedback configuration of the MPC controller corresponding to mode 0. In order to achieve fast sampling, (which requires a QP problem of moderate complexity) while maintaining sufficiently long prediction and control horizons, the blocking factor 2 was specified for both horizons.

The result of the simulation when the MPC controller is applied to the nonlinear plant (14) is shown in Figure 9. Reference trajectories specifying step sequences for the elevation and rotation angles respectively are applied to the system. As

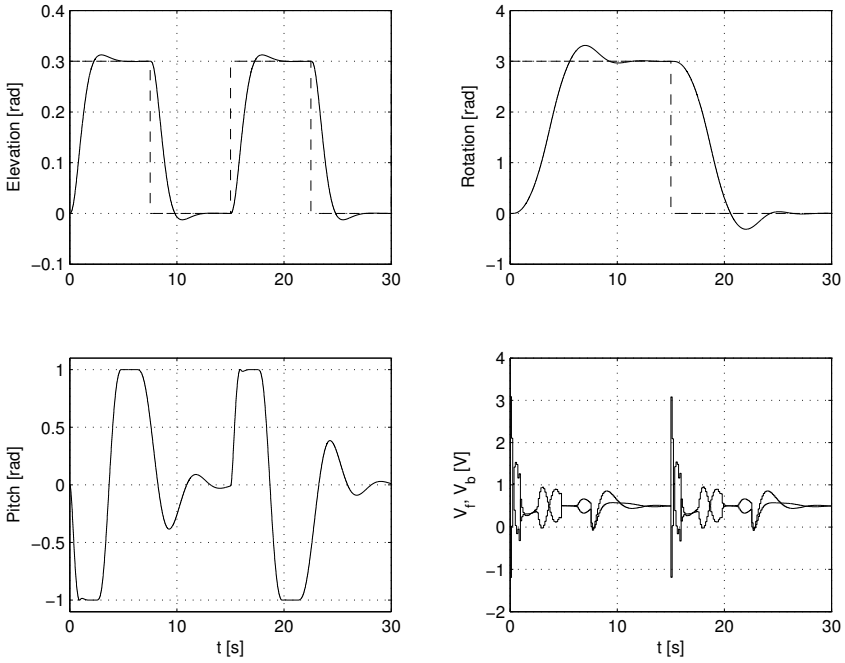

Figure 9 A simulation of the MPC controller applied to the nonlinear helicopter plant.

we can see, the controller manages to track the set points, while keeping the pitch angle within the specified limits.

8. MPCTools Command Reference

MPCinit

Purpose

Initializes the MPC data structure.

Syntax

```
md = MPCInit(Ad, Bd, Cyd, Czd, Dzd, Ccd, Dcd, Hp,
             Hw, zblk, Hu, ublk, du_max, du_min,
             u_max, u_min, z_max, z_min, Q, R, W, V,
             h, cmode, solver)
```

Input arguments

`Ad, Bd, Cyd`	System matrices for the plant; A_d, B_d and C_d.
`Czd, Dzd`	Matrices defining the controlled outputs; C_z and D_z.
`Ccd, Dcd`	Matrices defining the constrained outputs; C_c and D_c.
`Hw, Hp, Hu`	Integers defining the prediction and control horizons; H_w, H_p and H_u.
`zblk, ublk`	Blocking factors defining the sets I_p and I_u.
`u_min, u_max`	Control variable limits; u_{min} and u_{max}.
`du_min, du_max`	Control increment limits; Δu_{min} and Δu_{max}.
`z_min, z_max`	Controlled variable limits; z_{min} and z_{max}.
`Q, R`	Weighting matrices for the cost function.
`W, V`	Weighting matrices for the Kalman filter design, if applicable.
`h`	Sampling interval.
`cmode`	Controller mode.
`solver`	Solver to be used for the quadratic programming problem.

Output arguments

`md` Contains the pre-computed matrices needed by the MPC controller.

Description

`MPCInit` creates the data structure used by the MPC controller. A discrete time model, with sampling interval h, of the controlled system is assumed,

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= C_y x(k) \\ z(k) &= C_z x(k) + D_z u(k) \\ z_c(k) &= C_c x(k) + D_c u(k) \end{aligned} \tag{15}$$

where $z(k)$ are the controlled outputs, $y(k)$ the measured outputs and $z_c(k)$ the constrained outputs. The constraints of the system are given by

$$\begin{aligned} \Delta u_{min} \le \Delta u(k) \le \Delta u_{max}, \quad k \in I_u \\ u_{min} \le u(k) \le u_{max}, \quad k \in I_u \\ z_{min} \le z_c(k) \le z_{max}, \quad k \in I_p \end{aligned} \tag{16}$$

where $\Delta u(k) = u(k) - u(k-1)$ are the control increments. I_p and I_u are the sets of samples for which the controlled and control variables are included in the cost function and for which the constraints are enforced. The first sample to be included in the optimization procedure is H_w, and the total number of samples in the prediction horizon is H_p. The entries in the array `z_blk` indicates the time distance between two consecutive predicted samples. Also, the last entry in `z_blk` is used as distance between the remaining samples (if `length(z_blk)` $<$ H_p).

Example

Assume that $H_w = 1$, $H_p = 6$ and `z_blk` = [1 2 2 5]. Then the samples [1 2 4 6 11 16] will be included in the cost function evaluation.

Equivalently, H_u indicates the number of predicted control moves to be calculated at each sample. The array `u_blk` contains the blocking factors indicating over which sampling intervals the control signal should be constant. For example, a blocking factor of 2 indicates that 2 consecutive control signals are equal.

Example

Assume that $H_u = 3$, `u_blk` = [1 2]. Then it is assumed that at each sample, for the predicted control signal, $\hat{u}$, we have that $\hat{u}(k+1) = \hat{u}(k+2)$ and $\hat{u}(k+3) = \hat{u}(k+4)$. Only $\hat{u}(k)$, $\hat{u}(k+1)$ and $\hat{u}(k+3)$ will be calculated. Q and R are weighting matrixes for the cost function, where Q penalizes the controlled outputs and R penalizes control increments. W and V are weighting matrices for the design of the Kalman filter used to estimate the state and load disturbances. W is the covariance matrix of the state and V is the covariance matrix of the measurement noise.

The controller supports several control modes. `cmode` specifies which mode should be used. The following modes are supported:

- `Mode 0`: State feedback
  The arguments W and V are not used, and may be set to `[]`. C_y is assumed to be identity.
- `Mode 1`: State feedback and explicit integrators
  The integrators act the controlled outputs, specified by C_z. The Q-matrix should be extended to include weights for the integrator states as well as the controlled outputs. The arguments W and V are not used, and may be set to `[]`.
- `Mode 2`: Observer based output feedback
  The design of the Kalman filter is based on the covariance matrices W and V.
- `Mode 3`: Observer based output feedback with explicit integrators
  The integrators act on the controlled outputs, specified by C_z. The Q-matrix should be extended to include weights for the integrator states as well as the controlled outputs. The controlled variables should be the same as the first p_z (p_z = `dim(z)`) measured variables.
- `Mode 4`: Disturbance observer based output feedback
  The disturbance model ensures that the controller achieves error free tracking. Constant load disturbances are assumed on the control input, and the

system is augmented to include also the disturbance states. If the number of measured outputs exceed the number of inputs, constant load disturbances on the additional measured outputs are assumed. The controlled variables should be the same as the first p_z measured variables. Also, the number of controlled outputs should be equal to the number of inputs.

The argument solver should be a string containing one of the alternatives `qp_as`, `qp_ip` or `quadprog`, indicating which solver should be used to solve the quadratic programming problem. Notice that `quadprog` requires Optimization Toolbox.

See Also

`MPCOptimizeSol`, `MPCSim` and `MPCfrsp`.

MPCOptimizeSol

Purpose

This function solves the MPC optimization problem.

Syntax

```
[duPred, zPred, J] = MPCOptimizeSol(x_est,u_last,du_old,
                                    r,md)
```

Input arguments

`x_est`	The current estimate of the state vector
`u_last`	The control signal applied to the plant at the last sample.
`du_old`	The optimal decision vector from the last sample that is used to hot start the quadratic programming algorithm.
`r`	The reference vector for the controlled variables.
`md`	The data structure containing pre-computed matrices.

Output arguments

`duPred`	Optimal control increment trajectory.
`zPred`	Optimal trajectory of the controlled variables.
`J`	Optimal value of the cost function.

Description

`MPCOptimizeSol` solves the MPC optimization problem. `duPred` and `zPred` are the predicted control increments and controlled outputs for the specified prediction horizons.

An estimate of the current state vector is given by `x_est`, and `u_last` is the last applied control input. As an initial starting point for the optimization algorithm, the last predicted control input vector, `du_old`, is supplied. `r` is the desired set point for the controlled outputs and md is the data structure containing matrices needed to solve the optimization problem.

See Also

`MPCInit`, `MPCSim` and `MPCfrsp`

MPCSim

Purpose

Simulates a linear system model controlled by the MPC controller.

Syntax

```
[x, u, y, z, zPredTraj, uPredTraj] = MPCSim(md,r,d)
```

Input arguments

`md`	Data object containing the pre-computed matrices.
`r`	Reference trajectory for the controlled variables.
`d`	Input disturbance trajectory.

Output arguments

`x`	State trajectory of the plant.
`u`	Control variable trajectory.
`y`	Measured variable trajectory.
`z`	Controlled variable trajectory.
`zPredTraj`	Optimal predicted trajectories for the controlled variables at each sample.
`uPredTraj`	Optimal predicted trajectories for the control variables at each sample.

Description

`MPCSim` simulates the MPC controller specified by the data object `md`. The reference trajectory for the controlled outputs is given by `r` and a load disturbance acting on the input is given by `d`.

See Also

`MPCInit`, `MPCOptimizeSol` and `MPCfrsp`

MPCController

Purpose

S-function for simulation of the MPC controller in the Simulink environment.

Syntax

```
[sys, x0, str, ts] = MPCController(t, x, u, flag, md)
```

Input arguments

`t`	Supplied by the Simulink environment.
`x`	Supplied by the Simulink environment.
`u`	Supplied by the Simulink environment.
`flag`	Supplied by the Simulink environment.
`md`	Data object containing the pre-computed matrices needed by the MPC controller. Supplied as a parameter in the S-function block.

Output arguments

`sys`	Needed by the Simulink environment.
`x0`	Needed by the Simulink environment.
`str`	Needed by the Simulink environment.
`ts`	Needed by the Simulink environment.

Description

`MPCController` is an S-function implementing the MPC controller intended for use with Simulink. The argument md, which is the only user supplied argument, contains the data structures needed by the controller. The input to the S-function block is a vector signal consisting of the measured outputs and the reference values for the controlled outputs. The output of the S-function block is a vector signal consisting of the control variables and the estimated state vector, potentially including estimated disturbance states.

See Also

`MPCOptimizeSol` and `MPCInit`

MPCfrsp

Purpose

Calculates the linear controller corresponding to the MPC controller if no constraints are active.

```
[Sys_CL, Sys_S, Sys_CS, Sys_SU, F, H, K, h] = MPCfrsp(md)
```

Input arguments

`md`	MPC data object.
`fignbr`	If this argument `fignbr is given, two plots with numbers fignbr and fignbr+1 are drawn.`

Output arguments

`Sys_CL`	Closed loop system from r to z.
`Sys_S`	Sensitivity function from v to y.
`Sys_CS`	Complimentary sensitivity function: from n to y.
`Sys_SU`	Control signal sensitivity function: from n to u.
`F`	See block diagram.
`H`	See block diagram.
`K`	See block diagram.
`h`	A vector of handles to the figures where the transfer functions are plotted.

See Figure 10 for explanation of the notation.

Description

`MPCfrsp` calculates the frequency responses of the MPC controller. When no constraints are active, the MPC controller is a linear controller and may be analyzed using linear methods. Some important transfer functions are also plotted.

See Also

`MPCInit`

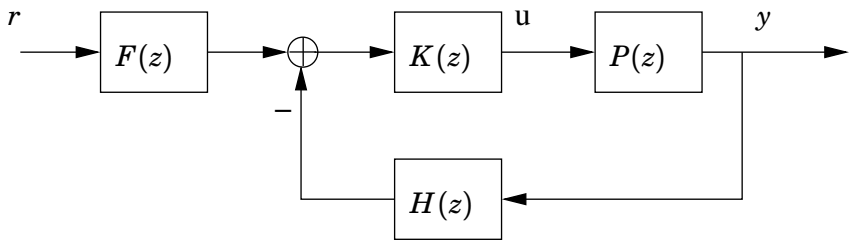

Figure 10 Block diagram used for calculation of the linear controller.

qp_as

Purpose

Solves a quadratic program using an active set method.

Syntax

```
[xopt, lambda, J, x_hist] = qp_as(H,f,A,b,x0)
```

Input arguments

`H`	H-matrix (Hessian) of the QP problem.
`f`	f-vector of the QP problem.
`A`	A-matrix defining linear constraints.
`b`	b-vector defining linear constraints.
`x0`	Initial solution guess.

Output arguments

`xopt`	The optimal solution.
`lambda`	Lagrange multipliers.
`J`	The optimal value of the cost function.
`x_hist`	History of x during the optimization run. Each column in `x_hist` represents the solution at the corresponding iteration.

Description

`qp_as` solves the quadratic programming problem

$$\min \frac{1}{2}x^T Hx + f^T x$$
$$\text{s.t.}$$
$$Ax \leq b$$

The algorithm is based on [Fletcher, 1987, p. 240].

See Also

`get feasible`

qp_ip

Purpose

Solves a quadratic program using a primal-dual interior point method.

Syntax

```
[xopt, lambda, J, x_hist] = qp_ip(H,f,A,b,x0)
```

Input arguments

`H`	H-matrix (Hessian) of the QP problem.
`f`	f-vector of the QP problem.
`A`	A-matrix defining linear constraints.
`b`	b-vector defining linear constraints.
`x0`	Initial solution guess.

Output arguments

`xopt`	The optimal solution.
`lambda`	Lagrange multipliers.
`J`	The optimal value of the cost function.
`x_hist`	History of x during the optimization run. Each column in `x_hist` represents the solution at the corresponding iteration.

Description

`qp_ip` solves the quadratic programming problem

$$\min \frac{1}{2} x^T H x + f^T x$$

s.t.

$$Ax \leq b$$

The algorithm is based on [Wright, 1997]. An initial solution is supplied in `x0`, but is not used efficiently by the algorithm in the current implementation.

getfeasible

Purpose

Finds a feasible solution subject to linear inequality constraints.

Syntax

```
[x, as, iter] = getfeasible(A,b)
```

Input arguments

`A`	A-matrix defining linear constraints.
`b`	b-vector defining linear constraints.

Output arguments

`x`	A feasible solution.
`as`	The indices of active constraints, if any.
`iter`	The number of iterations.

Description

`getfeasible` finds a feasible vector that fulfills the linear inequality constraints

$$Ax \leq b.$$

The algorithm is based on [Fletcher, 1987, p. 166]

See Also

`qp_as`

9. References

Åkesson, J. and P. Hagander (2003): "Integral action—a disturbance observer approach." In *Proceedings of European Control Conference*.

Åström, K. J. and B. Wittenmark (1990): *Computer Controlled Systems—Theory and Design*. Prentice-Hall, Englewood Cliffs, New Jersey.

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.

Bazaraa, M. S., H. D. Sherall, and C. M. Shetty (1993): *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, Inc.

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.

Fletcher, R. (1987): *Practical Methods of Optimization*. John Wily & Sons Ltd.

Johansson, K. H. (1997): *Relay Feedback and Multivariable Control*. PhD thesis ISRN LUTFD2/TFRT--1048--SE, Department of Automatic Control, Lund Institute of Technology, Sweden.

Maciejowski, J. M. (2002): *Predictive Control with Constraints*. Pearson Education.

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**, pp. 733–764.

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

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