The MPC Simulink Library

A. Bemporad
M. Morari
N. L. Ricker

User's Guide
Version 1

How to Contact A. Bemporad:

Dept. of Information Engineering
University of Siena

Via Roma, 56

53100 Siena, Italy

http://www.dii.unisi.it/~bemporad/

bemporad@dii.unisi.it

The MPC Simulink Library

Printing History: Jan 2000 Draft

MPC Simulink Library

Preface

Disclaimer

State of Development

The MPC Simulink Library is in a developmental (beta) stage. *It is not an official MathWorks product*. It has been tested extensively, but it is likely that some problems remain. Please review all calculations with a critical eye.

We will also be improving features in response to your comments.

In Case of Difficulty

This beta version of the MPC Simulink Library *is not being supported by The MathWorks*. If you encounter a problem, please check the FAQ (see "Updates" below). If that doesn't help, please send a detailed description by e-mail to A. Bemporad at the following address:

bemporad@dii.unisi.it

He would also be happy to have your suggestions for improvements in the usability of the MPC Simulink Library, its documentation, and the MPC Toolbox in general.

Updates

The FAQ and latest beta version of the MPC Simulink Library will be maintained for download at

http://www.dii.unisi.it/~bemporad/toolbox/mpclib.html

Acknowledgements

We appreciate the help of Federica Rusconi, Domenico Mignone, Carles Pedret Ferre, Adrian Toller, Konrad Stadler, Tobias Raithel, Kazuro Tsuda, Jay Lee, Pascal Gahinet, and Greg Wolodkin.

1

Introduction

MPC Simulink Library Overview

The MPC Simulink Library is designed to help you analyze and simulate Model Predictive Control (MPC) modules within any Simulink description of the environment.

Options

The MPC Simulink Library supports four controller blocks, to be connected in feedback with the system to regulate. Each block receives output measurements and returns the control input action to the system. The four blocks are the following:

- *Regulator*. The MPC block regulates the output of the system to zero.
- *Controller.* An additional reference signal is received by the MPC controller. The output of the system will track such a signal.
- *Controller with Measured Disturbance Rejection.* An additional information about measured input disturbance signals entering the system is received by the MPC controller and taken into account in the computation of the control action.
- *Controller with Anticipation.* Reference and measured disturbance signals are read from file. This allows knowing future samples when computing the control action.

Starting the MPC Simulink Library

System Requirements

You'll need the following MATLAB software:

- MPC Toolbox for MATLAB Version 5, including the MPC Simulink Library files (which are currently an add-on to the standard MPC Tools release).
- MATLAB Version 5.3 or greater.
- Simulink Version 3.0 or greater.
- Control Toolbox.

Installation

1. Make a backup copy of your MPC Tools directory (which should be located in your MATLAB directory, *e.g.*, `MATLAB/toolbox/mpc`). This will allow you to restore your original state if the GUI installation interferes in some unexpected way.

2. Download the latest version of the MPC Simulink Library from

   `http://control.ethz.ch/~bemporad/toolbox/mpclib.html`

   It's a ZIP archive. Extract the contents using one of the many available utility programs (e.g., WinZip). This should create a new `mpclib` directory within the `mpc` directory.

IMPORTANT: The ZIP archive is not a full version of MPC Tools. You must have existing `mpc/mpccmds` and `mpc/mpcdemos` directories. The files from the archive only update these directories.

---

3. Set your MATLAB Path so the new `mpc/mpclib` directory is included. Verify that the original `mpc/mpccmds` and `mpc/mpcdemos` directories are also on the path.

4. Type "`mpclib`" in the MATLAB Command window. You should see the MPC Simulink Library.

5. If it doesn't work or you have other problems, see section In Case of Difficulty in the preface to this document.

Quick Start

1. Type "`mpclibdemo`" in the command window.
2. Choose one of the tutorial examples.
3. Examine the Simulink MDL file and the Matlab M file for the demo.
4. Try it with your own applications!

Leisurely Start

The following sections of this document provide tutorial examples and additional details. If possible, work through the steps in MATLAB/Simulink as you read.

MPC Fundamentals

This section reviews MPC concepts and defines key terms. If you're already familiar with MPC you may prefer to read Chapter 2, "MPC Worksheet Basics" next.

Overview: single-input, single-output (SISO) MPC

Figure 1-1 shows a situation in which MPC is trying to hold a single variable, $\bar{y}$, at a target value, *r*, by adjusting the "manipulated variable" (or "actuator") *u*. See Table 1-1 for a brief description of the signals appearing in Figure 1-1.

The box labeled "plant" is the real process or mechanism that produces $\bar{y}$. It responds to changes in *u* as well as to two types of disturbance signals: measured, *v*, and unmeasured, *d*. (Some applications have unmeasured disturbances only; the *v* signal is optional.)

The unmeasured disturbance represents the usual mysterious events that upset plant operation, causing variations in $\bar{y}$. The only sign that such an event has occured is a change in the measured output, *y*.

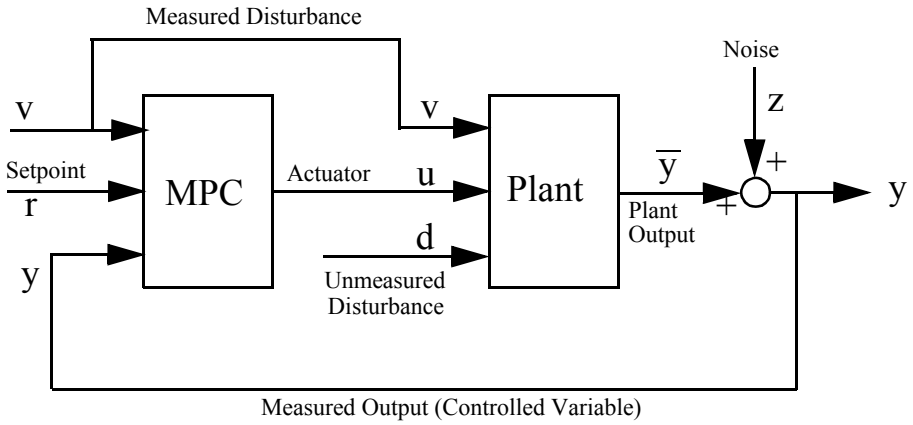


Figure 1-1 Block diagram of a single-input, single-output MPC application.

Table 1-1 Summary of MPC signals (see also Figure 1-1).

Symbol	Description
d	*Unmeasured disturbance*. A disturbance unknown to MPC that affects the plant output. MPC provides *feedback* compensation for such disturbances.
r	*Setpoint (or reference signal)*. The target value for the controlled variable.
u	*Actuator (or manipulated variable)*. The signal MPC adjusts in order to achieve its control objectives.
v	*Measured disturbance (optional)*. MPC *feedforward* compensation adjusts for such disturbances as they occur to minimize their impact on the controlled variable.
$\bar{y}$	*Controlled variable (or plant output)*. The signal to be held at the setpoint. This is the "true" value, uncorrupted by measurement noise.
y	*Measurement of the controlled variable*. Used to estimate the true value, $\bar{y}$.
z	*Measurement noise*. Represents electrical noise, sampling errors, drifting calibration, and other effects that reduce the accuracy of the measurement.

The measured disturbance also affects $\bar{y}$. The MPC block includes *models* of the way in which *v* and *u* affect $\bar{y}$ (symbolically, $v \to \bar{y}$ and $u \to \bar{y}$). It uses this information to calculate *u* adjustments that keep $\bar{y}$ at its setpoint in spite of the (known) disturbance. This calculation considers the effect of any known constraints on the adjustments (*e.g.*, an actuator at its upper or lower bound). If the models are accurate and the plant responds quickly to *u*, this "feedforward" compensation counteracts the effect of *v* perfectly.

NOTE: One may also specify bounds on $\bar{y}$. These constraint-handling properties are a distinguishing feature of MPC and can be particularly valuable when one has multiple control objectives to be achieved *via* multiple adjustments.

In reality, however, model imperfections, plant limitations, and unmeasured disturbances cause the measurement, *y*, to deviate from its expected value. Thus, MPC uses the output measurement and a "disturbance model" ($d \to \bar{y}$) to *predict* future changes in $\bar{y}$. It then uses its $u \to \bar{y}$ model to calculate appropriate adjustments (a form of "feedback" compensation). This calculation also considers the known constraints.

Various types of "noise" can corrupt the measurement. The signal *z* in Figure 1-1 represents such effects. They may vary randomly about a zero mean or exhibit a non-zero, drifting bias. MPC uses a $z \to y$ model in combination with its $d \to \bar{y}$ model to remove the estimated noise component of the measurement ("filtering").

The above feedforward/feedback actions are MPC's "regulator" mode. MPC also has a "servo" mode, *i.e.*, it adjusts *u* such that $\bar{y}$ tracks a time-varying setpoint. The tracking accuracy depends on the plant characteristics (including constraints), the accuracy of the $u \to \bar{y}$ model, and whether or not future setpoint variations can be *anticipated*, *i.e.*, known in advance.

Details: SISO case

Sampling period and sampling instants

MPC operates at discrete intervals of *Dt* time units, the "*sampling period*." Suppose that MPC starts at time $t = 0$. The "*sampling instants*" are the times at which MPC adjusts the manipulated variable, *u*. They are integer multiples of the sampling period: 0,*Dt*, 2*Dt*, 3*Dt*, ..., *kDt*, where the integer index, *k*, represents the current sampling instant.

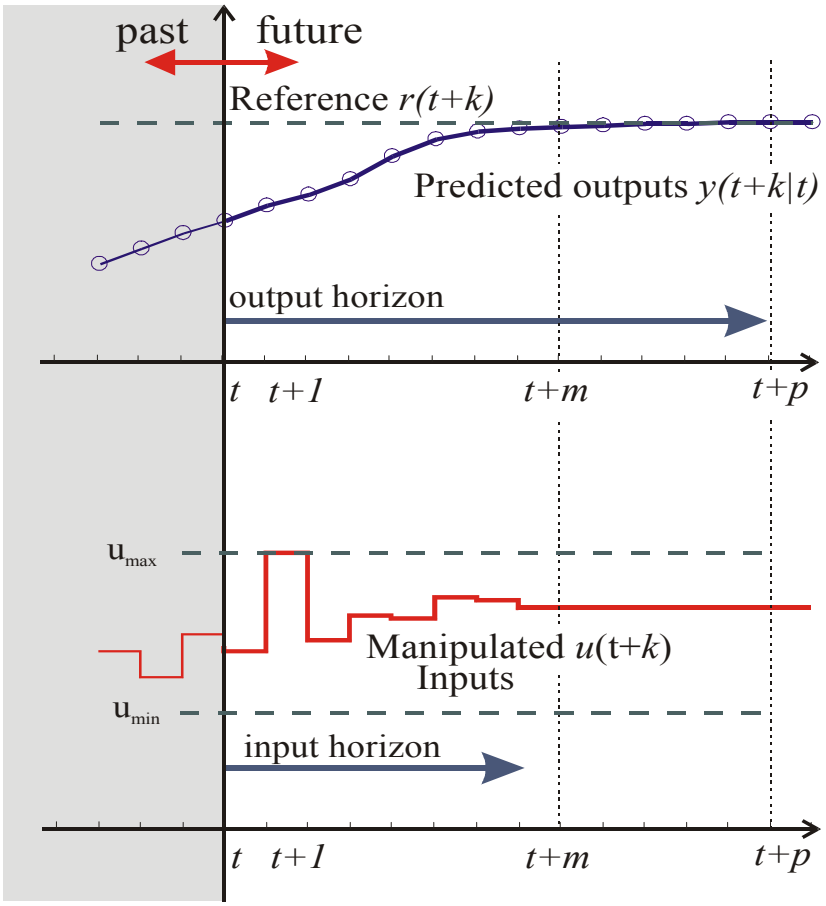


Figure 1-2 The MPC problem at the the sampling instant *t*.

An MPC sampling instant

Figure 1-2 shows the state of a hypothetical SISO MPC system which has been operating for some time; k is the current sampling instant. The current measured output, y_k, and previous measurements, $y_{k-1}, y_{k-2}, \ldots$, are known and are the filled circles in Figure 1-2 (a). The current measured disturbance, d_k, and its past values are also known (not shown).

Figure 1-2 (b) shows MPC's previous "moves," $u_{k-41}, ..., u_{k-1}$, as filled circles. As is usually the case, a "zero-order hold" receives each move from MPC and sends that value to the plant continuously until the next sampling instant – note the resulting step-wise variations in Figure 1-2 (b).

MPC must now calculate the current move, u_k. It does so in two phases:

1. *Estimation*. In order to make an intelligent move, MPC needs to know the current state of the system. This includes the true value of the controlled variable, $\bar{y}_k$, *and* any internal plant variables that influence the future trend, $\bar{y}_{k+1}, ..., \bar{y}_{k+P}$. To accomplish this MPC uses all past and current measurements and the models $u \to \bar{y}$, $d \to \bar{y}$, $w \to \bar{y}$, and $z \to y$. For details, see Chapter 4, "Disturbance Detection and Estimation".

2. *Optimization*. Values of setpoints, measured disturbances, and constraints are specified over a finite "horizon" of future sampling instants, $k+1$, $k+2$, ..., $k+P$, where P (a finite integer ≥ 1) is the "*prediction horizon*" – see Figure 1-2 (a). MPC then computes the M moves $u_k, u_{k+1}, ... u_{k+M-1}$, where M ($\geq 1, \leq P$) is the "*control horizon*" – see Figure 1-2 (b). In our hypothetical example, $P = 9$ and $M = 4$.

Suppose that the optimal sequence of moves is the series of four open circles in Figure 1-2 (b). MPC's model predicts that this will result in the output sequence shown as the nine open circles in Figure 1-2 (a).

These moves are *optimal* in the sense that no constraints are violated and the outputs track the setpoint "closely." See section Optimization for a detailed definition of optimality.

MPC sends only the move u_k to the plant. The plant operates with this input until MPC's next sampling instant, Dt time units later. MPC then obtains a new set of measurements and revises completely the plan it had formulated at the previous sampling instant, thus compensating for model error and unknown disturbances.

Prediction and control horizons

One might wonder why MPC bothers to optimize over P sampling periods into the future and calculate M moves when it discards all but the first move. Indeed, under certain conditions MPC gives identical results for $P = M = 1$ as it does for $P = M = \infty$. More often, however, the horizons have an important impact. Some examples are:

- *Constraints*. MPC's current move, u_k, depends on future constraints, *i.e.*, MPC plans ahead to avoid future constraint violations.
- *Delays in the plant*. Suppose that the plant includes a pure time delay; MPC's current move has no effect until D sampling periods into the future. Then we need $P \geq D$ and $M \leq P - D$ so that MPC can include the effect of each contemplated move in its optimization calculations.
- *Other nonminimum phase plants*. Consider a SISO plant with an inverse-response, *e.g.*, its unit-step response is a decrease below the initial condition followed by an increase to a final value above the initial condition.

Experience suggests the following rules of thumb:

1. Choose the MPC sampling period such that the plant's open-loop settling time is approximately 20-30 sampling periods (i.e., the sampling period is approximately one fifth of the dominant time constant).
2. Choose P to be the number of sampling periods used in step 1.
3. Use a relatively small M, *e.g.*, 3-5.

The resulting MPC performance will usually be insensitive to minor adjustments in these M and P values, and further "tuning" should not be necessary. If performance is poor, it is probably due to other factors, such as modeling errors.

Blocking

In Figure 1-2 (b), $M = 4$ and $P = 9$, and MPC is optimizing the first M moves of the prediction horizon, after which the manipulated variable remains constant for $P - M = 5$ sampling instants.

Figure 1-3 shows an alternative "blocked" strategy – again with $M = 4$ – in which the first planned move occurs at sampling instant k, the next at $k+2$, the next at $k+4$, and the final at $k+6$. A "block" is one or more successive sampling periods during which the manipulated variable is constant. The "block durations" are the number of sampling periods in each block. In Figure 1-3 the block durations are 2, 2, 2, and 3. (Their sum must equal P.)

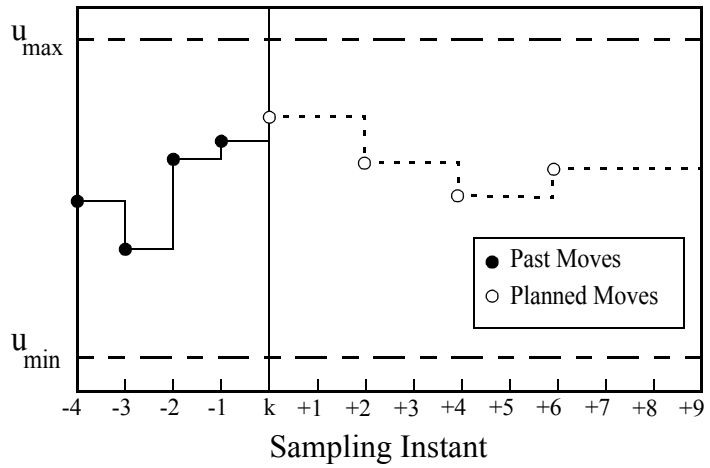


Figure 1-3 Example of blocking with M = [2, 2, 2, 3].

As for the default (unblocked) mode, only the current move, u_k, actually goes to the plant, and MPC repeats the optimization of *M* moves at its next sampling instant. Thus, the signal going to the plant actually varies every *Dt* time units.

So why use blocking? When $P >> M$ (as is generally recommended), and all *M* moves are at the beginning of the horizon, the moves tend to be larger (because all but the final move last just one sampling period). Blocking often leads to smoother adjustments, all other things being equal.

Optimization

For a SISO plant, MPC determines its moves by solving the following optimization problem (formulated for the k^{th} sampling instant):

$$\underset{u_k, \ldots, u_{k+P-1}}{Min} \sum_{i=1}^{P} [w^y (r_{k+i} - \hat{y}_{k+i})^2 + w^u (\Delta u_{k+i-1})^2] \qquad \textbf{(1-1)}$$

such that

$$\hat{y}_{k+i} = f(u_k, \ldots, u_{k+i-1}) \qquad \textbf{(1-2)}$$

$$u_{min} \le u_{k+i} \le u_{max} \quad \text{for i = 0:P-1} \tag{1-3}$$

$$y_{min} \le \hat{y}_{k+i} \le y_{max} \quad \text{for i = 1:P} \tag{1-4}$$

$$\left|\Delta u_{k+i}\right| \le \Delta u_{max} \quad \text{for i = 0:P-1} \tag{1-5}$$

where $\Delta u_j = u_j - u_{j-1}$ is the adjustment at sampling instant j, and w^y and w^u are non-negative "weights."

Equation (1-2) represents the *model* MPC uses to predict the controlled variable at instant $k+i$. In standard MPC it is a *linear function* of the adjustments $u_k, \ldots, u_{k+i-1}$.

Note: The prediction $\hat{y}_{k+i}$ is also a function of known and estimated disturbances, but these effects are constant, *i.e.*, independent of the adjustments. They are implicit in Equation (1-2).

Equations (1-3) – (1-5) are optional. They specify the bounds on the manipulated variable, the controlled variable, and the change in the manipulated variable.

When $P > M$ (the usual case), there are also $P - M$ equality constraints of the form $u_{k+j} = u_{k+j-1}$, *i.e.*, to hold the manipulated variables constant for a specified sequence of sampling instants. This depends on the specified block durations. For example, the situation of Figure 1-3 would have $u_{k+1} = u_k$, $u_{k+3} = u_{k+2}$, `etc`.

Choosing the optimization weights

The main difficulty in an MPC design (other than modeling) is the selection of the weights w^y and w^u in Equation (1-1). Some trial-and-error tuning is often needed. The following guidelines may be helpful:

- A positive value of w^y penalizes predicted deviations of the controlled variable from its setpoint. The larger this weight, the more closely MPC tries to track the setpoint. If $w^y = 0$, MPC ignores the setpoint.
- A positive value of w^u penalizes planned changes in the manipulated variable. A moderate value makes the controller "cautious," which usually increases its "robustness" (making it less sensitive to modeling errors – a positive effect). A large value makes it "sluggish," however. In the extreme as w^u goes to infinity, MPC ignores deviations of the controlled variable from its setpoint.

- In a SISO problem, only the ratio of w^y to w^u matters. A good strategy is to fix one of them at unity and vary the other, testing the performance in simulations to obtain the desired tradeoff between fast setpoint tracking (large ratio) and cautious adjustments of the manipulated variable (small ratio). (By default, the MPC GUI sets $w^y = 1, w^u = 0.1$.)
- MPC does whatever it must in order to satisfy any "hard" inequality constraints, equations (1-3)–(1-5). The values of w^y and w^u become less important when constraints are encountered.

Model Predictive Control Simulink Library

Alberto Bemporad, Manfred Morari, Larry Ricker

December 13, 2001

This document describes a new Simulink library for the Model Predictive Control Toolbox. Please report any bug or comment to `bemporad@dii.unisi.it`.

1 Description

The new MPC Simulink library contains four blocks, see Fig. 1. The MPC blocks are based on the model shown in Fig. 2. This consist of the LTI system[1]

$$\left\{ \begin{array}{rcl} x(k+1) & = & Ax(k) + B_u u(k) + B_v v(k) + B_d d(k) \\ y(k) & = & Cx(k) + D_v v(k) + D_d d(k) \end{array} \right. \tag{1}$$

where $x(k) \in \mathbb{R}^n$ represents the state of the system, $u(k)\mathbb{R}^{n_u}$ are manipulated variables (MV) or command inputs, $v(k)\mathbb{R}^{n_v}$ is a vector of measured disturbances (MD), $d(k) \in \mathbb{R}^{n_d}$ are unmeasured disturbances (UD), and $y(k) \in \mathbb{R}^{n_y}$ is the output vector, which is composed of measured outputs (MY) $y_m(k)$ and unmeasured outputs (UY) $y_u(k)$. Note that the matrix D_u is assumed to be zero, i.e. no direct feedthrough of MVs on the output vector. [2]

The unmeasured disturbance $d(k)$ is modeled as the output of the LTI system

$$\left\{ \begin{array}{rcl} x_d(k+1) & = & Fx_d(k) + Gn(k) \\ d(k) & = & Hx_d(k) + Kn(k) \end{array} \right. \tag{3}$$

System (3) is driven by the random Gaussian noise $n(k)$, which has zero mean and unit covariance matrix. For instance, a step-like unmeasured disturbance is modeled as the output of an integrator.

[1] If continuous time models are supplied, internally these are sampled with the controller's sampling time

[2] This assumption is justified by the following argument. Let T_s be the sampling time. The time τ_k required by the controller to compute $u(k)$ after $y_m(k) = y_m(kT_s)$ has been acquired, is always finite. Then, $u(t) \equiv u(k)$ for $kT_s + \tau_k \leq t < (k+1)T_s + \tau_{k+1}$. By assuming that zero-order holders are present in the control system, a direct feedthrough would give a contribution on $y_m(k)$ of the form $D_u u(kT_s) = D_u u(k-1)$. Therefore, the model can be described as

$$\left\{ \begin{array}{rcl} x(k+1) & = & Ax(k) + B_u u(k) + B_v v(k) + B_d d(k) \\ y(k) & = & Cx(k) + D_u u(k-1) + D_v v(k) + D_d d(k) \end{array} \right. \tag{2}$$

By extending the state $x(k)$ to include $u(k-1)$, one returns to the form (1).

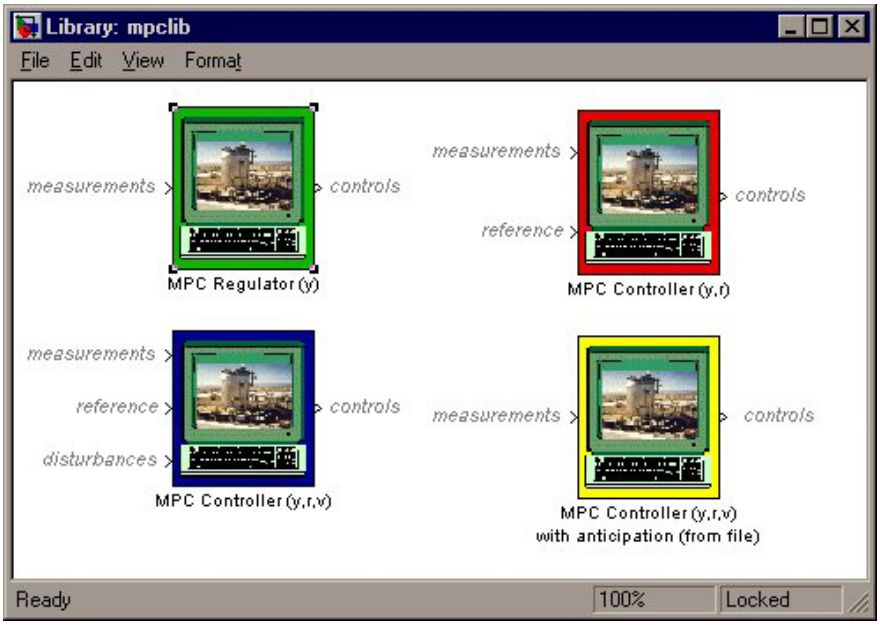


Figure 1: MPC Simulink Library.

The MPC controller selects the input $u(k)$ by solving the optimization problem

$$\left\{\begin{array}{l} \min\limits_{\Delta u(k|k),\ldots,\Delta u(m-1+k|k)} \sum\limits_{i=0}^{p-1} \left(\sum\limits_{j=1}^{n_y} |w^y_{i+1,j} \left[y_j(k+i+1|k) - r_j(k+i+1) \right]|^2 + \right. \\ \qquad \left. \sum\limits_{j=1}^{n_u} |w^{\Delta u}_{i,j} \Delta u_j(k+i|k)|^2 + \sum\limits_{j=1}^{n_u} |w^u_{i,j} \left[u_j(k+i|k) - u_{\text{target},j}(k) \right]|^2 \right) + \rho_\epsilon \epsilon^2 \\ \text{subj. to} \left\{ \begin{array}{rlcll} u_i^{\min} & -\epsilon V_i^{\text{u,min}} & \leq & u(k+i|k) & \leq u_i^{\max} \quad +\epsilon V_i^{\text{u,max}} \\ \Delta u_i^{\min} & -\epsilon V_i^{\Delta\text{u,min}} & \leq & \Delta u(k+i|k) & \leq \Delta u_i^{\max} \quad +\epsilon V_i^{\Delta\text{u,max}}, \quad i = 0,\ldots,p-1 \\ y_i^{\min} & -\epsilon V_i^{\text{y,min}} & \leq & y(k+i+1|k) & \leq y_i^{\max} \quad +\epsilon V_i^{\text{y,max}} \\ & \Delta u(k+j|k) & = & 0, & \qquad\qquad\qquad\qquad j = m,\ldots,p \\ & \epsilon & \geq & 0 & \end{array} \right. \end{array} \right. \tag{4}$$

with respect to the sequence of input increments $\{\Delta u(k|k), \ldots, \Delta u(m-1+k|k)\}$ and the slack variable ϵ. In (4), $*(k+i|k)$ denotes the value predicted for time $k+i$ based on the information available at time k; $r(k)$ is the current sample of the output reference. While only the measured outputs are connected to the MPC Simulink block, $r(k)$ is a reference *for all* the outputs (measured and unmeasured). When the reference r is not known in advance, the current reference $r(k)$ is used over the whole prediction horizon, namely $r(k+i+1) \equiv r(k)$ in (4). The exploitation of future references in MPC is referred to as *anticipative action*, and is only possible when the reference is read from file during simulation. A similar anticipative action can be performed with respect to measured disturbances $v(k)$, namely $v(k+i) = v(k)$ if the measured disturbance is not known in

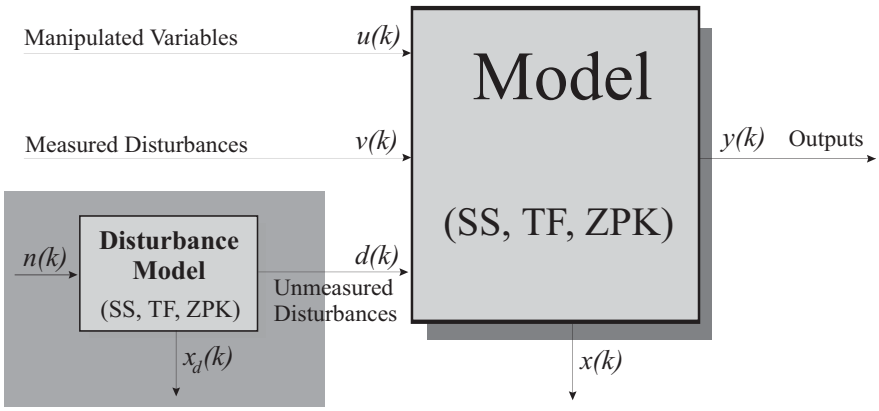


Figure 2: Model used by the MPC block for prediction/state estimation.

advance (e.g. is coming from another Simulink block) or $v(k+i)$ is obtained from a file. In the prediction, $d(k+i)$ is instead obtained by setting $n(k+i) \equiv 0$ in (3).

In (4), the constraints on u, Δu, and y are relaxed by introducing the slack variable $\epsilon \geq 0$. The vectors $V^{\mathrm{u,min}}$, $V^{\mathrm{u,max}}$ $V^{\Delta\mathrm{u,min}}$, $V^{\Delta\mathrm{u,max}}$ $V^{\mathrm{y,min}}$, and $V^{\mathrm{y,max}}$ represent *violation multipliers*, and vary continuously between 0 and 1. When $V_i^{\mathrm{u,max}} = 0$, the upper bound $u_i \leq u_{\max}$ becomes a *hard* constraint (otherwise is called *soft* constraint, as it may be violated). By default, input and input-variation constraints are treated as hard constraints ($V^{\mathrm{u,min}} = V^{\mathrm{u,max}} = V^{\Delta\mathrm{u,min}} = V^{\Delta\mathrm{u,max}} = 0$), while output constraints are considered as soft constraints with $V^{\mathrm{y,min}} = V^{\mathrm{y,max}} = 1$. Hard output constraints ($V^{\mathrm{y,min}} = 0$ or $V^{\mathrm{y,max}} = 0$) may lead the MPC controller to get stuck because of infeasibility of the optimization problem. Infeasibility might in fact occur when model and plant mismatch significantly, or just because of numerical round off. To prevent infeasibility of the MPC problem, a warning message is produced if $V^{\mathrm{y,min}}$ and $V^{\mathrm{y,max}}$ are smaller than 10^{-6}, and automatically adjusted at that value. By default, $\rho_\epsilon = 10^5 \max_{i=0}^{p-1}\{w_i^u, w_i^{\Delta u}, w_{i+1}^y\}$.

$u_{\mathrm{target}}(k)$ is a set-point for the input vector; [3] w_i^y, w_i^u, $w_i^{\Delta u}$ are nonnegative row vectors of weight coefficients; and $u_i^{\min}$, $u_i^{\max}$, $\Delta u_i^{\min}$, $\Delta u_i^{\max}$, $y_i^{\min}$, $y_i^{\max}$ are vectors of lower/upper bounds.

Only $\Delta u(k|k)$ is actually used to compute $u(k)$. The remaining samples $\Delta u(k+i|k)$ are discarded, and a new optimization problem based on $y_m(k+1)$ is solved at the next sampling step $k+1$.

As the true states $x(k)$, $x_d(k)$ are not available to the controller, predictions are obtained from the state estimates $\hat{x}(k)$, $\hat{x}_d(k)$. These are computed from the measured output $y_m(k)$ by the linear state observer

$$\begin{bmatrix} \hat{x}(k|k) \\ \hat{x}_d(k|k) \end{bmatrix} = \begin{bmatrix} \hat{x}(k|k-1) \\ \hat{x}_d(k|k-1) \end{bmatrix} + L\left[y_m(k) - C_m\hat{x}(k|k-1) - D_{vm}v(k) - D_{dm}H\hat{x}_d(k|k-1)\right]$$

where $*_m$ denotes the rows of C, D relative to measured outputs, and

$$\begin{aligned} \hat{x}(k+1|k) &= A\hat{x}(k|k) + B_u u(k) + B_v v(k) + B_d H\hat{x}_d(k|k) \\ \hat{x}_d(k+1|k) &= F x_d(k|k) \end{aligned}$$

[3] One typically uses u_{target} if the number of inputs is greater than the number of outputs, as a lower-priority set-point.

To prevent numerical difficulties in the absence of unmeasured disturbances, the gain L is designed as the Kalman gain for the extended model

$$\begin{aligned}\begin{bmatrix} x(k+1) \\ x_d(k+1) \end{bmatrix} &= \begin{bmatrix} A & B_d H \\ 0 & F \end{bmatrix} \begin{bmatrix} x(k) \\ x_d(k) \end{bmatrix} + \begin{bmatrix} B_u \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} B_v \\ 0 \end{bmatrix} v(k) + \begin{bmatrix} B_d K \\ G \end{bmatrix} n(k) + I\xi(k) \quad (5) \\ y_m(k) &= \begin{bmatrix} C_m & D_{dm} H \end{bmatrix} \begin{bmatrix} x(k) \\ x_d(k) \end{bmatrix} + D_{vm} v(k) + D_{dm} K n(k) + \zeta(k)\end{aligned}$$

where ξ, ζ are additional unmeasured disturbances having covariance matrix S_1 and S_2 respectively, the covariance matrix of n is assumed to be I, and ξ, ζ, n are considered independent. Note that $d(k)$ models both state disturbances and output disturbances ($B_d = 0$, $D_d \neq 0$).

Table 1: MPC blocks.

Block	**Description**	**Inputs**
MPC Regulator	Regulate the output to the origin	$y_m(t)$
MPC Controller	Track an arbitrary output set-point $r(t)$	$y_m(t)$, $r(t)$
MPC Controller+MD	Track an arbitrary output set-point $r(t)$ and reject a measured input disturbance $v(t)$	$y_m(t)$, $r(t)$, $v(t)$
MPC Controller+Anticipation	Same as MPC Controller+MV, but $r(t)$, $v(t)$ are read from a .MAT file	$y_m(t)$

The algorithm implemented in the MPC block uses different procedures depending on the presence of constraints. If all the bounds are infinite, then the problem (4) is solved analytically, otherwise a Quadratic Programming (QP) solver is used. The matrices associated with the quadratic optimization problem are reported in Appendix A.

Since soft output constraints are used, the QP problem should never be infeasible. For robustness, in case this should happen for some reason, the previous optimal sequence is applied, i.e. $u(k+1) = u(k) + \Delta u(k+1|k)$, and a warning message is given.

2 Parameters and Dialog Box

The parameters of the MPC block are entered through the mask reported in Fig. 3 by double-clicking the MPC block, and are described in Table 2.

Table 2: MPC block parameters.

Parameter	**Description**	**Type**	**Default**
`model`	LTI model used for predictions	LTI object	mandatory
`p`	prediction horizon	Real	`10`
`moves`	(blocking) moves	Real or Array	`p`
`limits`	upper and lower bounds on u,Δu, y	Cell Array of Arrays	`Inf`
`weights`	weights on $u - u_{\text{target}}$, Δu, $y - r$	Cell Array of Arrays	`{0,.1,1}`
`filename`	`MAT` file containing r, v	String	`"`
`MPCadd`	additional MPC parameters	Structure	`[]`

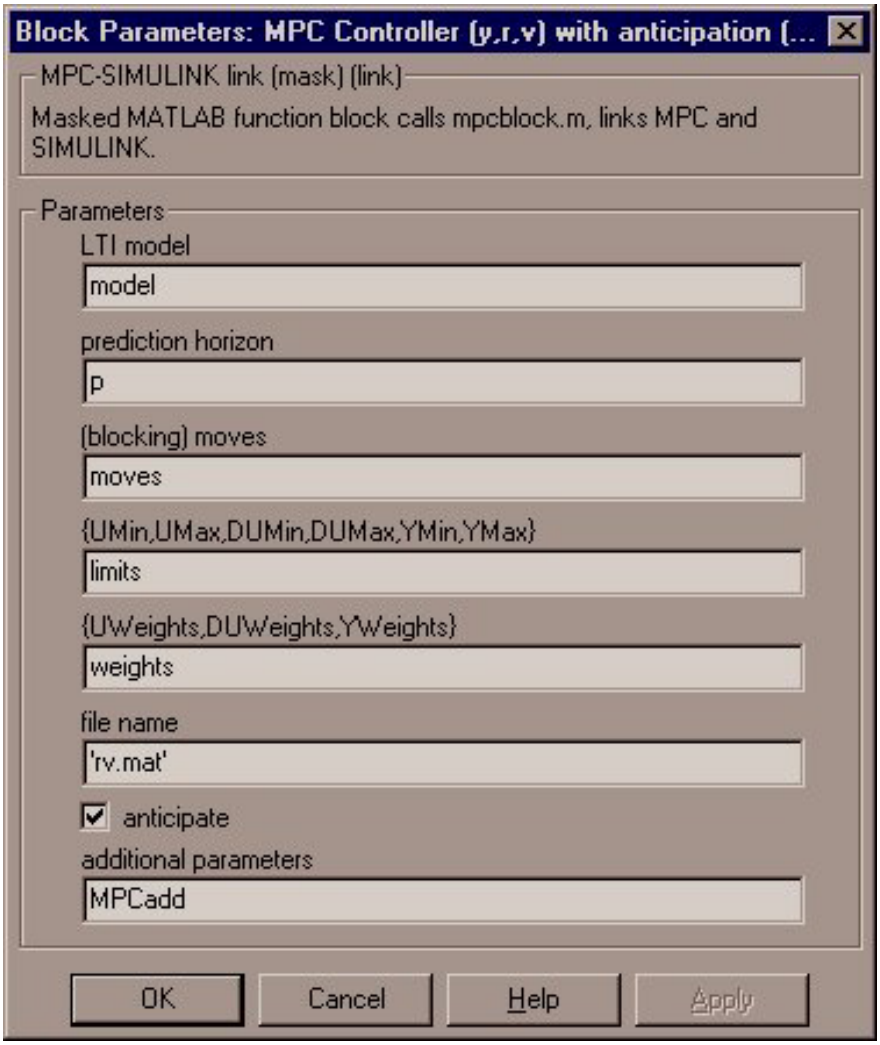


Figure 3: Mask of MPC block.

- `model` is an LTI object which is used for predictions and Kalman filtering, where `model.b` collects B_u, B_v, B_d (in general the columns can be scrambled, see `mvindex`, `mdindex` below), and similarly `model.d` collects $D_u = 0$, D_v, D_d.
- `p`≥ 1 is the prediction horizon. The case `p=Inf` is not implemented yet.
- `moves` is either a number $m \leq$`p` of free moves (i.e. $\Delta u(k+i|k) = 0$ for $i = m, \ldots,$`p`), or a vector of blocking moves (see e.g. `CMPC.M` in the MPC Toolbox Reference Guide or Appendix A).
- `limits` is a cell array of the form {`Umin,Umax,DUmin,DUmax,Ymin,Ymax`}, where `Umin[i,:]` $= (u_i^{\min})'$ (and similarly for the other matrices, cfr. `ulim` in `CMPC.M`).

- `weights` is a cell array of the form {`Uweight,DUweight,Yweight`}, where `Uweight[i,:]` $= w_i^u$ (and similarly for the other matrices) [4].

- `filename` is only present in the MPC block with anticipation, and specifies the `MAT` file where the reference $r(t)$ and the measured disturbance $v(t)$ signals are stored. The format is the same as in the `From File` Simulink block. The first variable saved in the `MAT` file is loaded from disk and is used to build the reference and measured disturbance signals. The variable is a matrix whose first row is a vector of times t, the following n_y rows are the reference $r(t)$, and the following n_v rows are the measured disturbance $v(t)$. Missing rows are treated as zeros. The signals are resampled with the MPC controller sampling time and stored in the MPC block memory. The first (last) sample is used for simulation time before (after) the specified range of t. Note that the same `MAT` file should be used for generating the actual signals r, v in the simulation, so that the MPC block has a consistent information (see Figure 6).

- `MPCadd` is a structure containing the fields indicated in Table 3.

Table 3: Structure of additional MPC parameters.

Field	**Description**	**Type**	**Default**
`utarget`	set-point for manipulated variables	Array	`0`
`mvindex`	indices of manip. vars (within input vector)	Array	`[1:`n_u`]`
`mdindex`	indices of meas. disturb. (within input vector)	Array	`[]`
`myindex`	indices of meas. outputs (within output vector)	Array	`[1:`n_y`]`
`noisemodel`	LTI model for unmeasured disturbances	Array	`[]`
`covmat`	estimator covariance matrices	Cell Array of Arrays	{}
`rhoeps`	weight on slack variable ρ_ϵ	Real	$10^5 \max_{i=0}^{p-1}\{w_i^u, w_i^{\Delta u}, w_{i+1}^y\}$
`violation`	violation multipliers $\{V^{\mathrm{u,min}}, \ldots, V^{\mathrm{y,max}}\}$	Cell	$\{0,0,0,0,1,1\}$
`x0`	initial state estimate $\hat{x}(0)$	Array	`zeros`
`u1`	initial past input $u(-1)$	Array	`zeros`
`ts`	controller sampling time	Real	`model.Ts`

  - `MPCadd.utarget` is a row vector of set-points for the input vector, as described above.
  - `MPCadd.mvindex` and `MPCadd.mdindex` are row vectors indicating the position of MV and MD variables (u, v) within the overall input vector, for instance B_u =`model.b(:,MPCadd.mvindex)`.
  - `MPCadd.myindex` is a row vector indicating the positions of MY variables y_m within the output vector y.
  - `MPCadd.noisemodel` is an LTI object which is used for estimating the unmeasured disturbance $d(k)$ (see (3) and Fig. 2). If empty, step-like unmeasured disturbances are assumed by default, i.e. one integrator for each unmeasured disturbance.
  - `MPCadd.covmat` is a cell array defining the covariance matrices S_1, S_2 for the additional disturbances $\xi(t)$, $\zeta(t)$ defined in (5). By default, if `MPCadd.covmat={}` then step distur-

[4] If `Umin`, `Uweight` have a number of rows smaller than `p`, the last row is automatically repeated (the same for the other limit and weight matrices)

bances are added on each output (i.e. integrators driven by white noise of unit covariance matrix), $S_1 = I$, $S_2 = I$ (if there are no unmeasured disturbances), or $S_1 = .001I$, $S_2 = .1I$ (if there are unmeasured disturbances). If `MPCadd.covmat={S1,S2}`, then `S1` (`S2`) can be either `[]` (default value), a scalar (S_1 =`S1`I), or a matrix.

- `MPCadd.rhoeps` is the weight ρ_ϵ on the slack variable ϵ representing the violation of the constraints. The larger ρ_ϵ with respect to input and output weights, the more the constraint violation is penalized. Although the same variable ϵ is used for softening all output constraints, different penalties can be induced by scaling the output components differently. For instance, if the constraint $y_1 > 0$ is much more important than the constraint $-1 \leq y_2 \leq 1$, the first row of the C and D matrices of the model can be multiplied by a large scalar, the constraints $y_1^{\min}$, $y_1^{\max}$ by the same scalar, and the output measurement and reference signal amplified by the same amount before being connected to the MPC controller, for consistency. Note that the output weights must be rescaled as well, accordingly.
- `MPCadd.violation` is a cell array of violation multipliers $\{V^{\mathrm{u,min}}\ V^{\mathrm{u,max}}$, $V^{\Delta\mathrm{u,min}}$, $V^{\Delta\mathrm{u,max}}$, $V^{\mathrm{y,min}}$, $V^{\mathrm{y,max}}\}$. As for the cell array `limits` and `weights`, the violation multipliers can be time-varying. By default, `MPCadd.violation`$=\{0,0,0,0,1,1\}$, corresponding to hard input and input-variation constraints on all inputs, and soft output constraints with equal degree of violation concern on all outputs.
- `MPCadd.x0` is the initial value for the state estimate $\hat{x}(0)$, and `u1` the initial value of the input u at time -1 (as $u(0) = \Delta u(0) + u(-1)$, nonzero values for `u1` might be needed if there are tight limits on Δu). The default is $\hat{x}(0) = 0$.
- `MPCadd.ts` is the sampling time of the controller: the MPC algorithm is executed every `MPCadd.ts` units of time. If `MPCadd.ts` is not given, then `MPCadd.ts` is automatically inherited from the sampling time `model.Ts`. If `MPCadd.ts`$\neq$`model.Ts`, then the model is resampled with sampling time `MPCadd.ts`.

3 Example 1

The script `EX1.M` prepares the parameters to simulate the Simulink model `MPC_EX1.MDL`, depicted in Fig. 4.

```
% 3x1 system
%
% MV=[1], MD=[2], UD=[3]
% Y=[1]

% Prepare parameters for mpc_ex1

% True plant & true initial state
sys=ss(tf({1,1,1},{[1 .5 1],[1 1],[.7 .5 1]}));
A=sys.a;
B=sys.b;
```

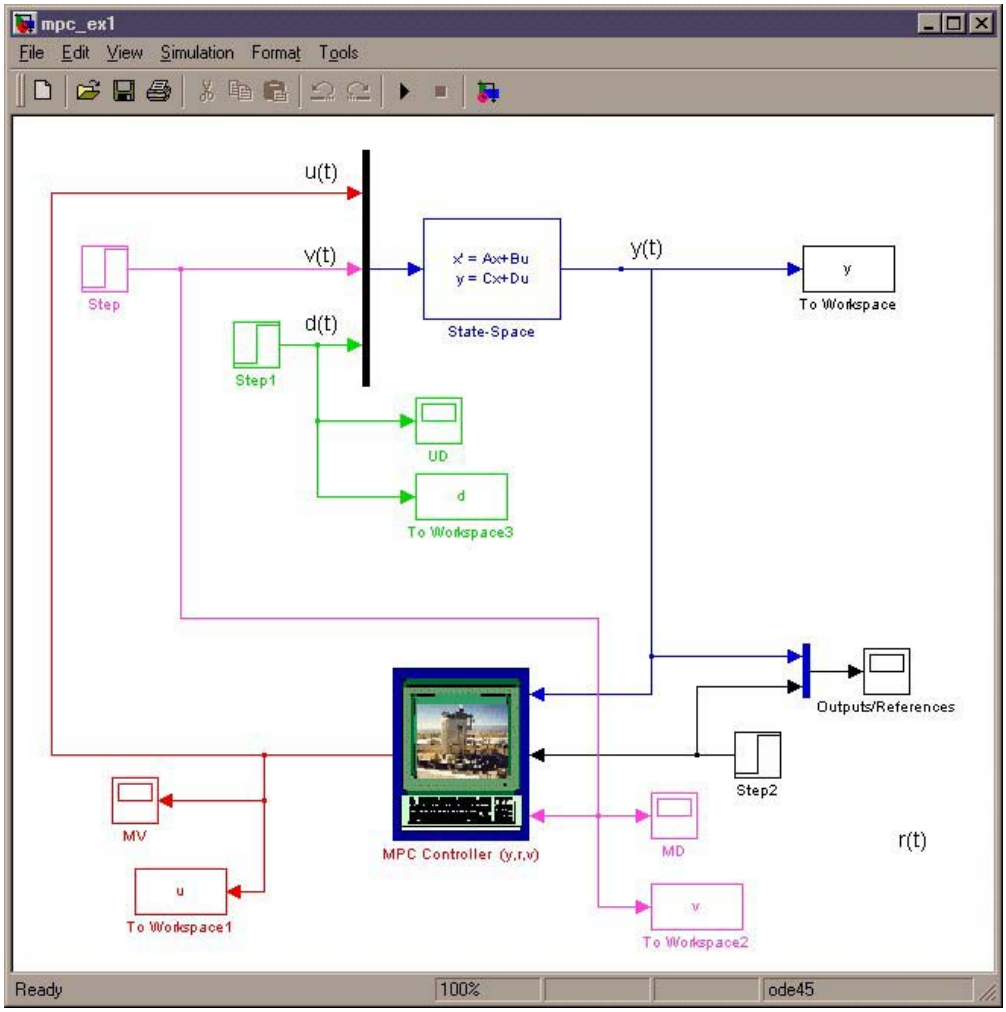


Figure 4: Simulink model `MPC_EX1.MDL`.

```
C=sys.c;
D=sys.d;

x0=[0 0 0 0 0]';


Ts=.2;      %Sample time
Tstop=30;   %Simulation time

% prediction model
model=c2d(sys,Ts);

p=10;
```

```
moves=3;

umin=0;
umax=1;
dumin=-Inf;
dumax=Inf;
ymin=-Inf;
ymax=Inf;

limits={umin,umax,dumin,dumax,ymin,ymax};
weights={[0],[.1],[1]};

clear MPCadd
MPCadd.mvindex=[1];
MPCadd.mdindex=[2];

open_system('mpc_ex1.mdl')
sim('mpc_ex1',Tstop)
```

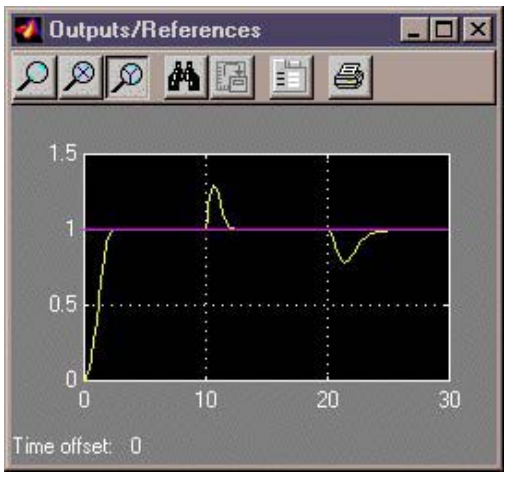
The LTI system `model` used for the MPC block is obtained by sampling the LTI continuous time system `sys`. This is also used as the plant to be controlled in Fig. 4. The reference trajectory $r(t)$ is a unit step. A step measured disturbance $v(t)$ of intensity 1 appears at time $t = 10$, while a step unmeasured disturbance $d(t)$ of intensity -0.5 appears at time $t = 20$. The input variable $u(t)$ is constrained within the range $[0, 1]$. The input and output trajectories are reported in Fig. 5.

3.1 Anticipative Action

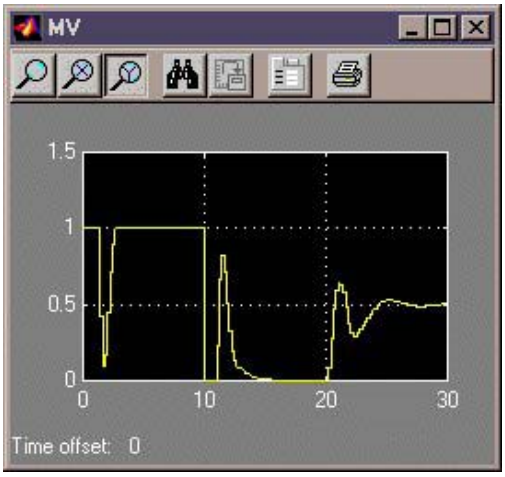
The script `EX1A.M` simulates the Simulink model `MPC_EX1A.MDL`, depicted in Fig. 6, which is a modified version of `MPC_EX1.MDL` where r and v signals are read from the `MAT` file `RV.MAT`, and MPC uses an anticipative action. Note that the same `MAT` file is used for generating the actual signals r, v in the simulation.

```
% Reference and measured disturbance signals
time=0:Ts:Tstop;
trv=[time;
   zeros(size(0:Ts:5)),ones(size(5+Ts:Ts:Tstop))
   zeros(size(0:Ts:10)),ones(size(10+Ts:Ts:Tstop))];

save rv trv
```

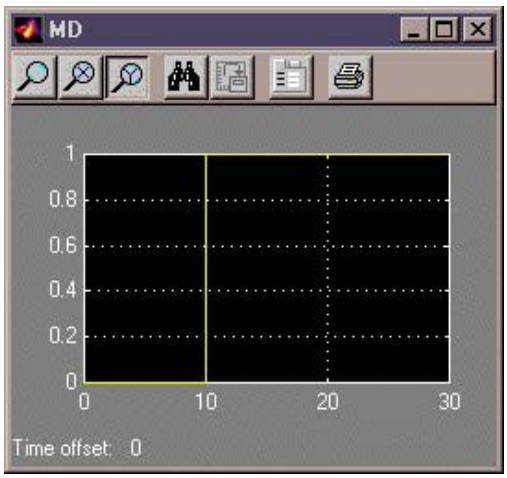
The input and output trajectories are reported in Fig. 7.



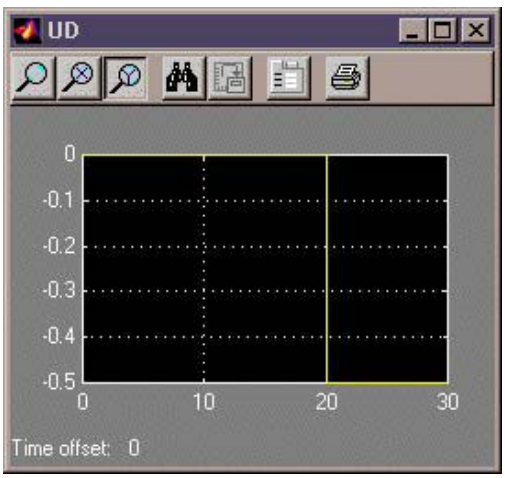
(a) Output $y(t)$ and reference $r(t)$.



(b) Manipulated variable $u(t)$.



(c) Measured disturbance $v(t)$.



(d) Unmeasured disturbance $d(t)$.

Figure 5: Trajectories obtained running `EX1.M`.

4 Example 2

The script `EX2NL.M` prepares the parameters to simulate the Simulink model `MPC_EX2NL.MDL`, depicted in Fig. 8.

```
% MIMO (3x2) SYSTEM: MV=[1 2 3], MY=[1 2]
%                     UD=[5 6]

% Prepare parameters for mpc_ex2nl
```

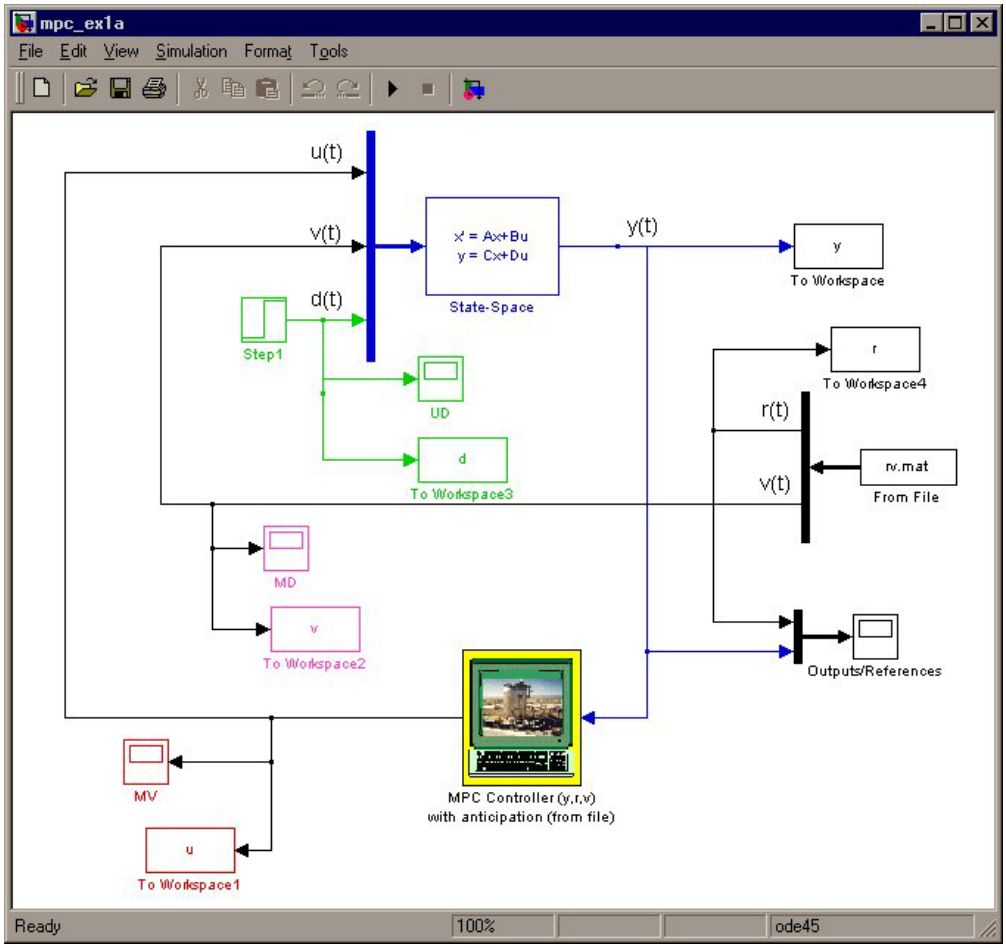
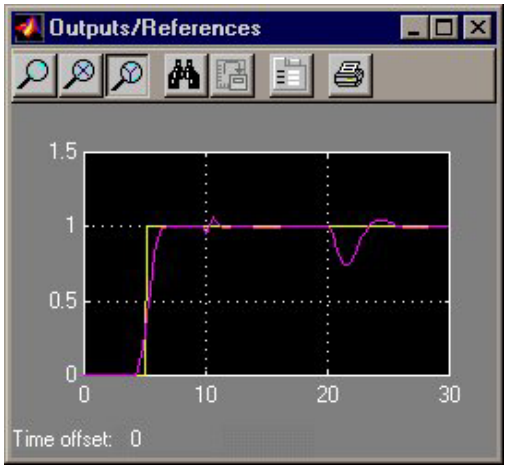
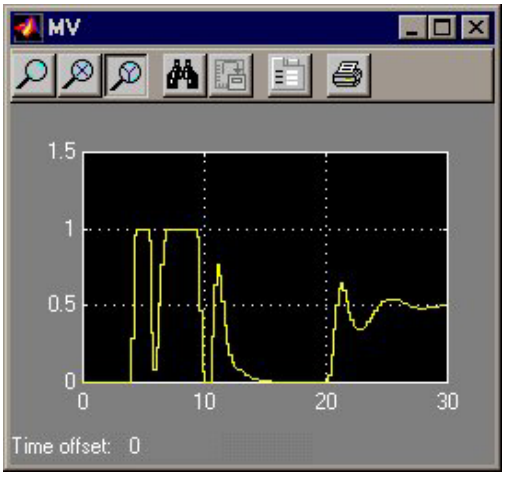


Figure 6: Simulink model `MPC_EX1A.MDL`.

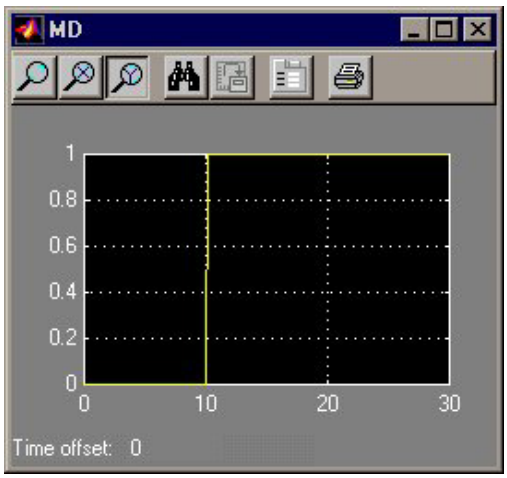
```
% Nonlinear model
sys='nl3x2';

% Linearize the model at (0,0)
[A,B,C,D]=linmod2(sys);

Ts=.2;                          %Sample time
model=c2d(ss(A,B,C,D),Ts);      %Convert to discrete time
clear A B C D

p=5;                            %Prediction horizon
moves=2;                        %Free control moves
```



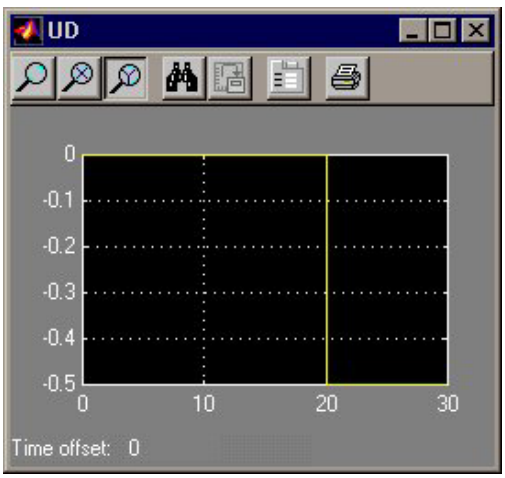
(a) Output $y(t)$ and reference $r(t)$.



(b) Manipulated variable $u(t)$.



(c) Measured disturbance $v(t)$.



(d) Unmeasured disturbance $d(t)$.

Figure 7: Trajectories obtained running `EX1A.M`.

```
% Define lower and upper bounds for u, delta u, y

umin=[-2,-1,-1];
umax=[2,1,1];
dumin=[-Inf,-Inf,-Inf];
dumax=[Inf,Inf,Inf];
ymin=[-Inf,-Inf];
ymax=[Inf,Inf];
```

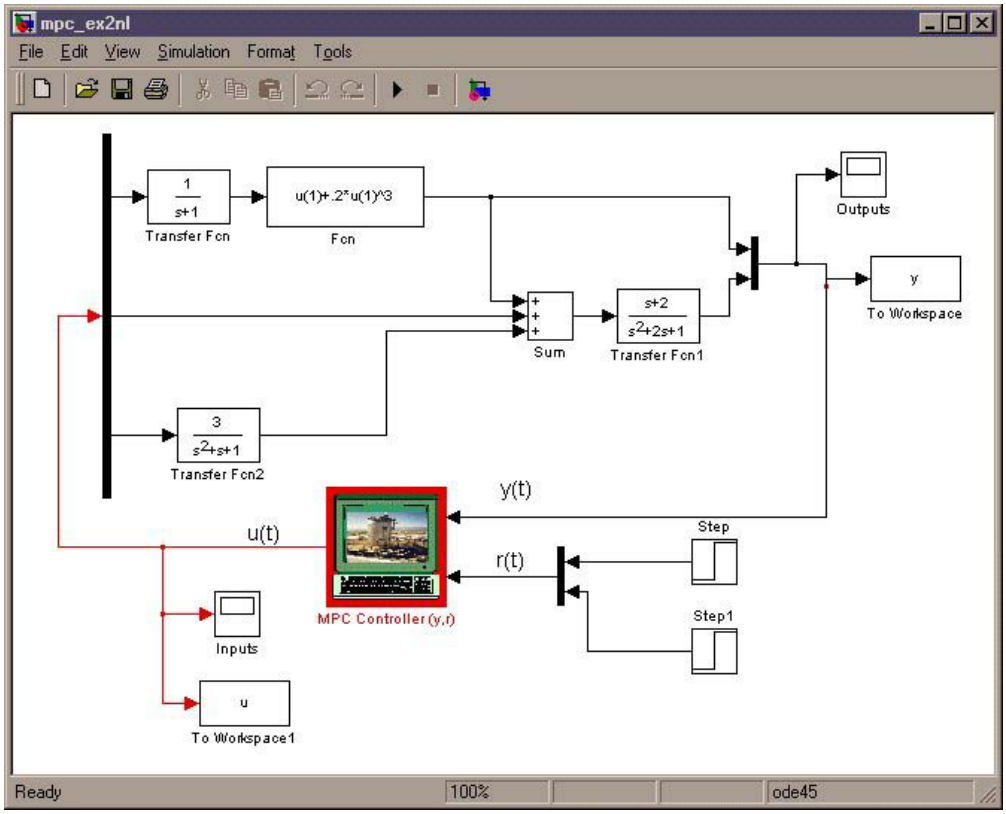


Figure 8: Simulink model `MPC_EX2NL.MDL`.

```
limits={umin,umax,dumin,dumax,ymin,ymax};

% Define weights for u, delta u, y

weights={[0 0 0],[.1 .1 .1],[1 1]};

% No noisemodel, no covmat supplied implies that
% output step disturbances will be added by default

open('mpc_ex2nl.mdl')              % Display the Simulink model

% Run simulation

Tfinal=8;
sim('mpc_ex2nl',Tfinal)
```

The LTI system `model` used for the MPC block is obtained by converting to discrete time the Simulink nonlinear model `NL3X2.MDL` depicted in Fig. 9. A copy of `NL3X2.MDL` is also used

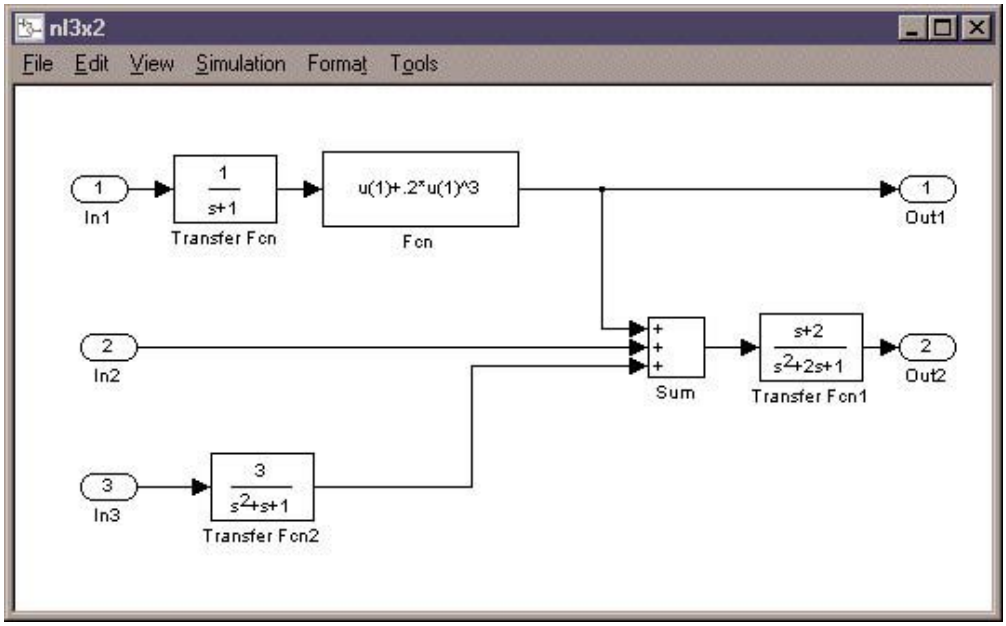


Figure 9: Simulink model `NL3X2.MDL`.

as the plant to be controlled in Fig. 8. In order to take into account steady-state offsets due to the nonlinearity, a step-like unmeasured disturbances is added on each output. The reference trajectories $r_1(t)$ and $r_2(t)$ are unit step, and $r_1(t)$ switches back to 0 at time $t = 4$. Constraints are present on the input variables $u_1(t)$, $u_2(t)$, $u_3(t)$. The input and output trajectories are reported in Fig. 10.

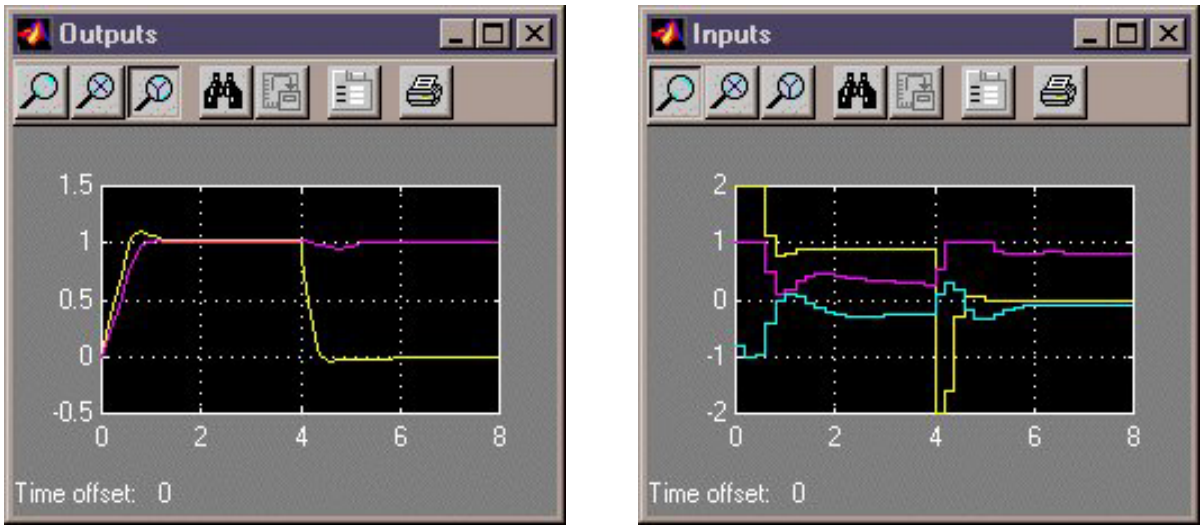
5 Upgrading from MPC Toolbox v1.0

- In the MPC Toolbox v1.0, the variable `uwt` refers to weights on *increments* of manipulated variables. Here, the corresponding variable has been called `DUWeight`.

6 Installation

The provided file archive must be extracted into a directory (e.g. `MPCBLOCK/`). This directory must be added to Matlab's path. The archive contains the following files:

```
mpc1.dll
mpc2.dll
mpcbl1.tif
mpcbl2.tif
mpcbl3.tif
mpcbl4.tif
mpclib.mdl
```



(a) Outputs.

(b) Inputs.

Figure 10: Trajectories obtained running `EX2NL.M`.

```
mpcsfun.m

Directory of \demos

ex1.m
ex10.m
ex11.m
ex12.m
ex1a.m
ex2nl.m
ex3.m
ex6.m
ex7.m
ex7u.m
ex9.m
mpc_ex1.mdl
mpc_ex10.mdl
mpc_ex11.mdl
mpc_ex12.mdl
mpc_ex1a.mdl
mpc_ex2nl.mdl
mpc_ex3.mdl
mpc_ex6.mdl
```

```
mpc_ex7.mdl
mpc_ex7u.mdl
mpc_ex9.mdl
nl3x2.mdl

Directory of \manual

mpcblock.pdf
mpcblock.ps
```

A Matrices for the MPC Quadratic Problem

This appendix describes the matrices built at initialization (routine `MPC1.M`) that will be used later (in the routine `MPC2.M`) for solving the MPC optimization problem.

Assume for simplicity that the disturbance model (3) is a unit gain ($d(k) = n(k)$ is white Gaussian noise). Then, the prediction model is given by

$$\left\{ \begin{array}{rcl} x(k+1) & = & Ax(k) + B_u u(k) + B_v v(k) + B_d n(k) \\ y(k) & = & Cx(k) + D_v v(k) + D_d n(k) \end{array} \right. \tag{6}$$

(if a model for unmeasured disturbances is specified, then (6) will be the augmented model (1)+(3)). Consider for simplicity the prediction at time $k = 0$. We set $n(i) = 0$ for all prediction instants i, and obtain

$$y(i|0) = C \left[A^i x(0) + \sum_{h=0}^{i-1} A^{i-1} B_u \underbrace{\left[u(-1) + \sum_{j=0}^{h} \Delta u(j) \right]}_{u(h)} + B_v v(k) \right] + D_v v(i) \tag{7}$$

which gives

$$\left[\begin{array}{c} y(1) \\ \vdots \\ y(p) \end{array} \right] = S_x x(0) + S_{u1} u(-1) + S_u \left[\begin{array}{c} \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{array} \right] + H_v \left[\begin{array}{c} v(0) \\ \vdots \\ v(p) \end{array} \right] \tag{8}$$

where

$$S_x = \left[\begin{array}{c} CA \\ CA^2 \\ \vdots \\ CA^p \end{array} \right] \in \mathbb{R}^{pn_y \times n_x}, \quad S_{u1} = \left[\begin{array}{c} CB_u \\ CB_u + CAB_u \\ \vdots \\ \sum_{h=0}^{p-1} CA^h B_u \end{array} \right] \in \mathbb{R}^{pn_y \times n_u}$$

$$S_u = \left[\begin{array}{cccc} CB_u & 0 & \dots & 0 \\ CB_u + CAB_u & CB_u & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{h=0}^{p-1} CA^h B_u & \sum_{h=0}^{p-2} CA^h B_u & \dots & CB_u \end{array} \right] \in \mathbb{R}^{pn_y \times n_u}$$

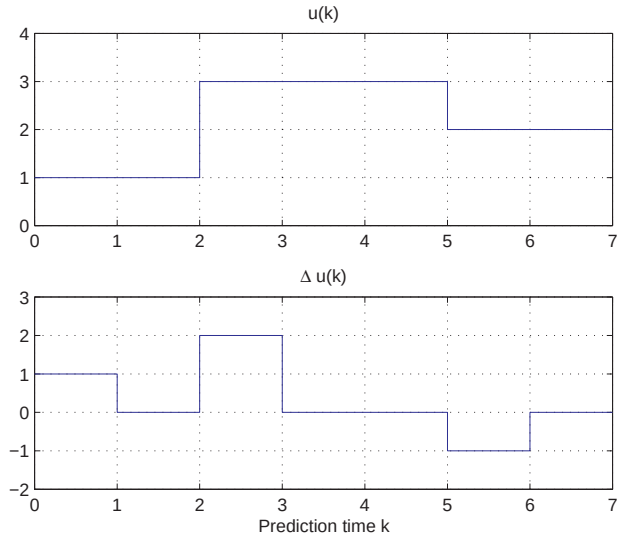


Figure 11: Blocking moves: inputs and input increments for `moves=[2 3 2]`

$$H_v = \begin{bmatrix} CB_v & D_v & 0 & \dots & 0 \\ CAB_v & CB_v & D_v & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{p-1}B_v & CA^{p-2}B_v & CA^{p-3}B_v & \dots & D_v \end{bmatrix} \in \mathbb{R}^{pn_y \times n_v}$$

Let m be the number of free moves, i.e., free optimization variables, that we denote by $z \triangleq [z_0, \dots, z_{m-1}]$. Then,

$$\begin{bmatrix} \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{bmatrix} = J_M \begin{bmatrix} z_0 \\ \vdots \\ z_{m-1} \end{bmatrix}$$

where $J_M \in \mathbb{R}^{(mn_u)\times(pn_u)}$ depends on the choice of blocking moves. Consider for instance the blocking moves depicted in Fig. 11, which corresponds to the choice `moves=[2 3 2]`, or, equivalently,

$$u(0) = u(1), \quad u(2) = u(3) = u(4), \quad u(5) = u(6)$$
$$\Delta u(0) = z_0, \ \Delta u(2) = z_1, \ , \Delta u(5) = z_2, \quad \Delta u(1) = \Delta u(3) = \Delta u(4) = \Delta u(6) = 0.$$

Then, the corresponding matrix J_M is

$$J_M = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix}$$

The file `MPC1.M` also defines the vector `DUFree`$\in \mathbb{R}^{mn_u}$, whose j-th entry is 1 if $j = kn_u + i$ and $\Delta u_i(k)$ is a free move (or, in other words, if the j-th row of J_M is nonzero).

The function to be optimized is

$$\begin{aligned} J \;=\; & \left(\begin{bmatrix} u(0) \\ \vdots \\ u(p-1) \end{bmatrix} - \begin{bmatrix} u_{\text{target}}(0) \\ \vdots \\ u_{\text{target}}(p-1) \end{bmatrix}\right)' \mathcal{W}_u^2 \left(\begin{bmatrix} u(0) \\ \vdots \\ u(p-1) \end{bmatrix} - \begin{bmatrix} u_{\text{target}}(0) \\ \vdots \\ u_{\text{target}}(p-1) \end{bmatrix}\right) + \\ & \begin{bmatrix} \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{bmatrix}' \mathcal{W}_{\Delta u}^2 \begin{bmatrix} \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{bmatrix} + \\ & \left(\begin{bmatrix} y(1) \\ \vdots \\ y(p) \end{bmatrix} - \begin{bmatrix} r(1) \\ \vdots \\ r(p) \end{bmatrix}\right)' \mathcal{W}_y^2 \left(\begin{bmatrix} y(1) \\ \vdots \\ y(p) \end{bmatrix} - \begin{bmatrix} r(1) \\ \vdots \\ r(p) \end{bmatrix}\right) + \rho_\epsilon \epsilon^2 \end{aligned} \tag{9}$$

where

$$\begin{aligned} \mathcal{W}_u &\triangleq \text{diag}\left(w^u_{0,1},\ w^u_{0,2},\ \ldots,\ w^u_{0,n_u},\ \ldots,\ w^u_{p-1,1},\ w^u_{p-1,2},\ \ldots,\ w^u_{p-1,n_u}\right), \\ \mathcal{W}_{\Delta u} &\triangleq \text{diag}\left(w^{\Delta u}_{0,1},\ w^{\Delta u}_{0,2},\ \ldots,\ w^{\Delta u}_{0,n_u},\ \ldots,\ w^{\Delta u}_{p-1,1},\ w^{\Delta u}_{p-1,2},\ \ldots,\ w^{\Delta u}_{p-1,n_u}\right), \\ \mathcal{W}_y &\triangleq \text{diag}\left(w^y_{1,1},\ w^y_{1,2},\ \ldots,\ w^y_{1,n_y},\ \ldots,\ w^y_{p,1},\ w^y_{p,2},\ \ldots,\ w^y_{p,n_y}\right) \end{aligned}$$

Finally, after substituting $u(k)$, $\Delta u(k)$, $y(k)$, J can be rewritten as

$$J = \rho_\epsilon \epsilon^2 + z' K_{du} z + 2 \left[\begin{bmatrix} r(1) \\ \vdots \\ r(p) \end{bmatrix}' K_r + \begin{bmatrix} v(0) \\ \vdots \\ v(p) \end{bmatrix}' K_v + u'(-1) K_u + u'_{\text{target}} K_{ut} + x'(0) K_x \right] z. \tag{10}$$

Let us now consider the limits on inputs, input increments, and outputs[5]:

$$\begin{bmatrix} y^{\min}(1) - \epsilon V^{\text{y,min}}(1) \\ \vdots \\ y^{\min}(p) - \epsilon V^{\text{y,min}}(p) \\ u^{\min}(0) - \epsilon V^{\text{u,min}}(0) \\ \vdots \\ u^{\min}(p-1) - \epsilon V^{\text{u,min}}(p-1) \\ \Delta u^{\min}(0) - \epsilon V^{\Delta\text{u,min}}(0) \\ \vdots \\ \Delta u^{\min}(p-1) - \epsilon V^{\Delta\text{u,min}}(p-1) \end{bmatrix} \leq \begin{bmatrix} y(1) \\ \vdots \\ y(p) \\ u(0) \\ \vdots \\ u(p-1) \\ \Delta u(0) \\ \vdots \\ \Delta u(p-1) \end{bmatrix} \leq \begin{bmatrix} y^{\max}(1) + \epsilon V^{\text{y,max}}(1) \\ \vdots \\ y^{\max}(p) + \epsilon V^{\text{y,max}}(p) \\ u^{\max}(0) + \epsilon V^{\text{u,max}}(0) \\ \vdots \\ u^{\max}(p-1) + \epsilon V^{\text{u,max}}(p-1) \\ \Delta u^{\max}(0) + \epsilon V^{\Delta\text{u,max}}(0) \\ \vdots \\ \Delta u^{\max}(p-1) + \epsilon V^{\Delta\text{u,max}}(p-1) \end{bmatrix}$$

[5]Limits that are not finite are removed, as well as the input and input-increment limits over blocked moves.

along with the constraint $\epsilon \geq 0$. Similarly to what was done for the cost function, we can substitute $u(k)$, $\Delta u(k)$, $y(k)$, and obtain

$$M_z z + M_\epsilon \epsilon \leq M_{\text{lim}} + M_v \begin{bmatrix} v(0) \\ \vdots \\ v(p) \end{bmatrix} + M_u u(-1) + M_x x(0) \tag{11}$$

where

$$\begin{aligned} M_{\text{lim}} &= [y^{\max}(1); y^{\min}(1); \ldots; y^{\max}(p); y^{\min}(p); u^{\max}(0); u^{\min}(0); \ldots; u^{\max}(p-1); \\ &\quad u^{\min}(p-1); \Delta u^{\max}(0); \Delta u^{\min}(0); \ldots; \Delta u^{\max}(p-1); \Delta u^{\min}(p-1)], \\ M_\epsilon &= -[V^{\text{y,max}}(1); V^{\text{y,min}}(1); \ldots; V^{\text{y,max}}(p); V^{\text{y,min}}(p); V^{\text{u,max}}(0); V^{\text{u,min}}(0); \ldots; V^{\text{u,max}}(p-1); \\ &\quad V^{\text{u,min}}(p-1); V^{\Delta\text{u,max}}(0); V^{\Delta\text{u,min}}(0); \ldots; V^{\Delta\text{u,max}}(p-1); V^{\Delta\text{u,min}}(p-1)]. \end{aligned}$$

B On-Line Solution to the Optimization Problem

This appendix describes how the MPC optimization problem is solved at each time step k (in the routine `MPC2.M`) by using the matrices built at initialization (routine `MPC1.M`).

B.1 Unconstrained MPC

The optimal solution is computed analytically:

$$z^* = -K_{du}^{-1} \left[\begin{bmatrix} r(1) \\ \vdots \\ r(p) \end{bmatrix}' K_r + \begin{bmatrix} v(0) \\ \vdots \\ v(p) \end{bmatrix}' K_v + u'(-1) K_u + u'_{\text{target}} K_{ut} + x'(0) K_x \right]'$$

and the MPC controller sets $\Delta u(k) = z_0$, $u(k) = u(k-1) + \Delta u(k)$.

B.2 Constrained MPC

The optimal solution z^*, ϵ^* is computed by solving the quadratic program (10)-(11), using the `DANTZGMP.M` routine.