Model Predictive Control Toolbox

For Use with MATLAB®

Alberto Bemporad
Manfred Morari
N. Lawrence Ricker

■ Computation

■ Visualization

■ Programming

User's Guide

Version 2

The MathWorks

How to Contact The MathWorks:

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	www.mathworks.com/contact_TS.html	Technical support
@	suggest@mathworks.com	Product enhancement suggestions
	bugs@mathworks.com	Bug reports
	doc@mathworks.com	Documentation error reports
	service@mathworks.com	Order status, license renewals, passcodes
	info@mathworks.com	Sales, pricing, and general information

☎ 508-647-7000 (Phone)

508-647-7001 (Fax)

✉ The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web

Revision History

Contents

<div style="text-align: right">

MPC Simulink Library

</div>

3

<div style="text-align: right">

Case-Study Examples

</div>

4

Design Tool

5

Function Reference

6

Block Reference

7

Object Reference

8

Index

1

Introduction

What Is the Model Predictive Control Toolbox?

The Model Predictive Control (MPC) Toolbox is a collection of software that helps you design, analyze, and implement an advanced industrial automation algorithm. Like other MATLAB® tools, it provides a convenient graphical user interface (GUI) as well as a flexible command syntax that supports customization.

As its name suggests, MPC automates a target system (the "plant") by combining a prediction and a control strategy. An approximate, linear plant model provides the prediction. The control strategy compares predicted plant states to a set of objectives, then adjusts available actuators to achieve the objectives while respecting the plant's *constraints*. Such constraints can include the actuators' physical limits, boundaries of safe operation, and lower limits for product quality.

MPC's constraint-tolerance differentiates it from other "optimal control" strategies (e.g., the Linear-Quadratic-Gaussian approach supported in the Control System Toolbox). The impetus for this is industrial experience suggesting that the drive for profitability often pushes the plant to one or more constraints. MPC's explicit consideration of such factors allows it to allocate the available plant resources intelligently as the system evolves over time.

The Model Predictive Control Toolbox uses the same powerful linear dynamic modeling tools found in the Control System Toolbox and the System Identification Toolbox. You can employ transfer functions, state-space matrices, or a combination of the two. You can also include delays, which are a common feature of industrial plants.

If you don't have a model but can perform experiments, you can use the System Identification Toolbox to develop a data-based model, then exploit it in the Model Predictive Control Toolbox.

If you'd rather use Simulink® graphical tools to model your plant, the Model Predictive Control Toolbox provides a Simulink block for that environment. For example, you can easily linearize a nonlinear Simulink plant, use the linearized model to build a Model Predictive Controller block, and evaluate its control of the nonlinear plant.

Finally, you can use MPC tools in Simulink to develop and test a control strategy, then implement it in a real plant using Real Time Workshop®.

Model Predictive Control of a SISO Plant

The usual Model Predictive Control Toolbox application involves a plant having multiple inputs and multiple outputs (a *MIMO* plant).

Consider instead the simpler application shown in Figure 1-1 (see summary of nomenclature in Table 1-1). This plant could be a manufacturing process, such as a unit operation in an oil refinery, or a device, such as an electric motor. The main objective is to hold a single *output*, $\bar{y}$, at a *reference value* (or *setpoint*), r, by adjusting a single *manipulated variable* (or *actuator*) u. This is what is generally termed a *SISO* (single-input single-output) plant. The block labeled MPC represents a Model Predictive Controller designed to achieve the control objective.

The SISO plant actually has multiple inputs, as shown in Figure 1-1. In addition to the manipulated variable input, u, there may be a measured disturbance, v, and an unmeasured disturbance, d.
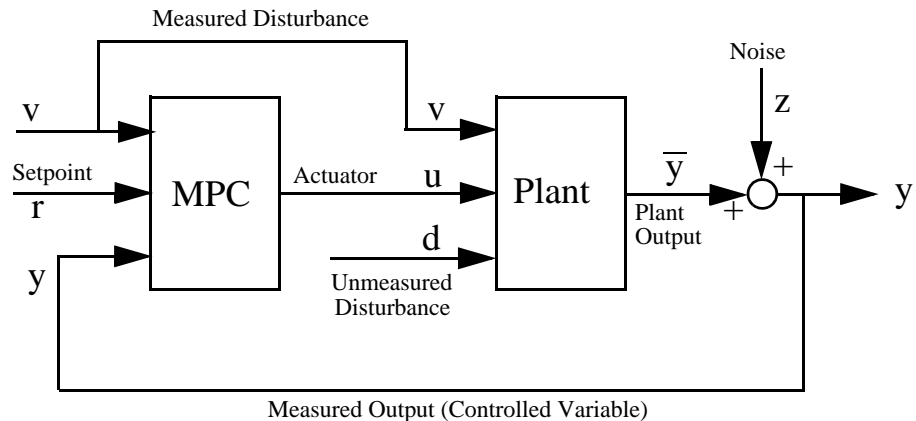
Figure 1-1: Block Diagram of a SISO Model Predictive Control Toolbox Application

The unmeasured disturbance is always present. As shown in Figure 1-1, it is an *independent* input – not affected by the controller or the plant. It represents all the unknown, unpredictable events that upset plant operation. (In the context of Model Predictive Control, it can also represent unmodeled dynamics.) When such an event occurs, the only indication is its effect on the *measured* output, y, which is *fed back* to the controller as shown in Figure 1-1.

Table 1-1: Description of Model Predictive Control Toolbox Signals

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

Some applications have unmeasured disturbances only. A *measured* disturbance, v, is another independent input affecting $\bar{y}$. In contrast to d, the controller receives the measured v directly, as shown in Figure 1-1 This allows the controller to compensate for v's impact on $\bar{y}$ immediately rather than waiting until the effect appears in the y measurement. This is called *feedforward* control.

In other words, a Model Predictive Control Toolbox design always provides *feeback* compensation for unmeasured disturbances and *feedforward* compensation for any measured disturbance.

The Model Predictive Control Toolbox design requires a *model* of the impact that v and u have on $\bar{y}$ (symbolically, $v \to \bar{y}$ and $u \to \bar{y}$). It uses this *plant model* to calculate the u adjustments needed to keep $\bar{y}$ at its setpoint.

This calculation considers the effect of any known constraints on the adjustments (typically an actuator upper or lower bound, or a constraint on how rapidly u can vary). One may also specify bounds on $\bar{y}$. These constraint specifications are a distinguishing feature of a Model Predictive Control Toolbox design and can be particularly valuable when one has multiple control objectives to be achieved *via* multiple adjustments (a MIMO plant). In the context of a SISO system, such contraint handling is often termed an *anti-windup* feature.

If the plant model is accurate, the plant responds quickly to adjustments in u, and no constraints are encountered, feedforward compensation can counteract the impact of v perfectly. In reality, model imperfections, physical limitations, and unmeasured disturbances cause the y to deviate from its setpoint. Therefore, the Model Predictive Control Toolbox design includes a *disturbance model* ($d \to \bar{y}$) to *estimate* d and *predict* its impact on $\bar{y}$. It then uses its $u \to \bar{y}$ model to calculate appropriate adjustments (feedback). This calculation also considers the known constraints.

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

The above feedforward/feedback actions comprise the controller's *regulator* mode. The Model Predictive Control Toolbox design also provides 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. If so, it provides feedforward compensation for these.

A Typical Sampling Instant

A Model Predictive Control Toolbox design generates a *discrete-time* controller – one that takes action at regularly spaced, discrete time instants. The *sampling instants* are the times at which the controller acts. The interval

separating successive sampling instants is the *sampling period*, Δt (also called the *control interval*). This section provides more details on the events occuring at each sampling instant.
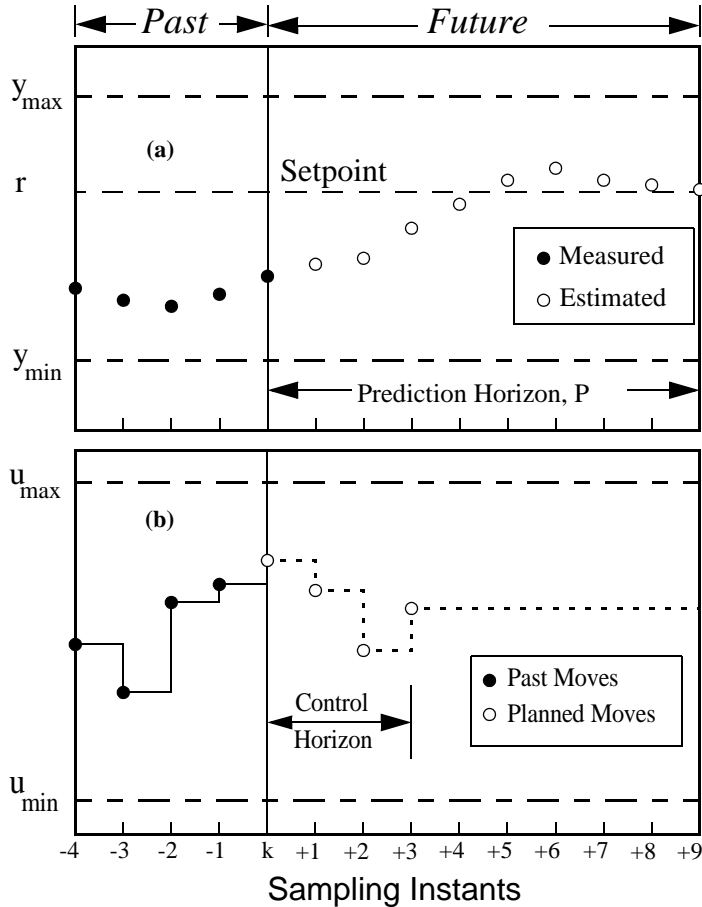
Figure 1-2: Controller State at the kth Sampling Instant

Figure 1-2 shows the state of a hypothetical SISO MPC system that has been operating for many sampling instants. Integer k represents the current instant. The latest measured output, y_k, and previous measurements, y_{k-1}, y_{k-2},

..., are known and are the filled circles in Figure 1-2(a). If there is a measured disturbance, its current and past values would be known (not shown).

Figure 1-2 (b) shows the controller'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 the controller and holds it until the next sampling instant, causing the step-wise variations shown in Figure 1-2 (b).

To calculate its next *move*, u_k the controller operates in two phases:

1 *Estimation*. In order to make an intelligent move, the controller needs to know the current state. This includes the true value of the controlled variable, $\bar{y}_k$, *and* any internal variables that influence the future trend, $\bar{y}_{k+1}$, ..., $\bar{y}_{k+P}$. To accomplish this, the controller 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 "Prediction" and "State 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). The controller computes *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 the hypothetical example shown in the figure, $P = 9$ and $M = 4$. The moves are the solution of a *constrained* optimization problem. For details of the formulation, see Chapter 2, "Optimization Problem".

In the example, the optimal moves are the four open circles in Figure 1-2 (b), and the controller predicts that the resulting output values will be the nine open circles in Figure 1-2 (a). Notice that both are within their *constraints*, $u_{min} \leq u_{k+j} \leq u_{max}$ and $y_{min} \leq y_{k+i} \leq y_{max}$.

When it's finished calculating, the controller sends move u_k to the plant. The plant operates with this *constant* input until the next sampling instant, Δt time units later. The controller then obtains new measurements and *totally revises its plan*. This cycle repeats indefinitely.

Reformulation at each sampling instant is essential for good control. The predictions made during the optimization stage are imperfect. Periodic measurement feedback allows the controller to correct for this error and for unexpected disturbances.

Prediction and Control Horizons

You might wonder why the controller bothers to optimize over P future sampling periods and calculate M future moves when it discards all but the first move in each cycle. Indeed, under certain conditions a controller using $P = M = 1$ would be identical to one using $P = M = \infty$. More often, however, the horizon values have an important impact. Some examples follow:

- *Constraints*. Given sufficiently long horizons, the controller can "see" a potential constraint and avoid it – or at least minimize its adverse effects. For example, consider the situation depicted below in which one controller objective is to keep plant output y below an upper bound y_{max}. The current sampling instant is k, and the model predicts the upward trend y_{k+i}. If the controller were looking P_1 steps ahead, it wouldn't be concerned by the constraint until more time had elapsed. If the prediction horizon were P_2, it would begin to take corrective action immediately.
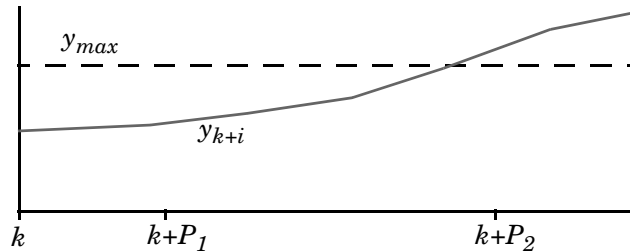
- *Plant delays*. Suppose that the plant includes a pure time delay equivalent to D sampling instants. In other words, the controller's current move, u_k, has no effect until y_{k+D+1}. In this situation it is essential that $P >> D$ and $M << P - D$, as this forces the controller to consider the full effect of each move.

  For example, suppose $D = 5$, $P = 7$, $M = 3$, the current time instant is k, and the three moves to be calculated are u_k, u_{k+1}, and u_{k+2}. Moves u_k, u_{k+1} would have some impact within the prediction horizon, but move u_{k+2} would have none until y_{k+8}, which is outside. Thus, u_{k+2} is indeterminant. Setting $P = 8$ (or $M = 2$) would allow a unique value to be determined. It would be better to increase P even more.

- *Other nonminimum phase plants*. Consider a SISO plant with an inverse-response, i.e., a plant with a short-term response in one direction,

but a longer term response in the opposite direction. The optimization should focus primarily on the longer-term behavior. Otherwise, the controller would move in the wrong direction.

Most designers choose P and M such that controller performance is insensitive to small adjustments in these horizons. Here are typical rules of thumb for a lag-dominant, stable process:

1 Choose the control interval 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 prediction horizon P to be the number of sampling periods used in step 1.

3 Use a relatively small control horizon M, *e.g.*, 3-5.

If performance is poor, you should examine other aspects of the optimization problem and/or check for inaccurate controller predictions.

MIMO Plants

One advantage of a Model Predictive Control Toolbox design (relative to classical multi-loop control) is that it generalizes directly to plants having multiple inputs and outputs. Moreover, the plant can be *non-square*, i.e., having an unequal number of actuators and outputs. Industrial applications involving hundreds of actuators and controller outputs have been reported.

The main challenge is to tune the controller to achieve multiple objectives. For example, if there are several outputs to be controlled, it might be necessary to prioritize so that the controller provides accurate setpoint tracking for the most important output, sacrificing others when necessary, e.g., when it encounters constraints. The Model Predictive Control Toolbox features support such prioritization.

Optimization and Constraints

As discussed in more detail in Chapter 2, "Optimization Problem", the Model Predictive Control Toolbox controller solves anoptimization problem much like the LQG optimal control described in the Control System Toolbox. The main difference is that the Model Predictive Control Toolbox optimization problem includes explicit *constraints* on u and y.

Setpoint Tracking

Consider first a case with no constraints. A primary control objective is to force the plant outputs to track their setpoints.

Specifically, the controller predicts how much each output will deviate from its setpoint within the prediction horizon. It multiplies each deviation by the output's weight, and computes the weighted sum of squared deviations, $S_y(k)$, as follows:

$$S_y(k) = \sum_{i=1}^{P} \sum_{j=1}^{n_y} \left\{ w^y_j [r_j(k+i) - y_j(k+i)] \right\}^2$$

where k is the current sampling interval, $k+i$ is a future sampling interval (within the prediction horizon), P is the prediction horizon, n_y is the number of plant outputs, w^y_j is the *weight* for output j, and $[r_j(k+i) - y_j(k+i)]$ is the predicted deviation at future instant $k+i$.

If $w_j^y \gg w_{i \neq j}^y$ the controller does its best to track r_j, sacrificing r_i tracking if necessary. If $w_j^y = 0$, on the other hand, the controller completely ignores deviations r_j–y_j.

Choosing the weights is a critical step. You will usually need to tune your controller, varying the weights to achieve the desired behavior.

As an example, consider Figure 1-3, which depicts a type of chemical reactor (a CSTR). Feed enters continuously with reactant concentration C_{Ai}. A reaction takes place inside the vessel at temperature T. Product exits continuously, and contains residual reactant at concentration C_A ($<C_{Ai}$).

The reaction liberates heat. A coolant having temperature T_c flows through coils immersed in the reactor to remove excess heat.
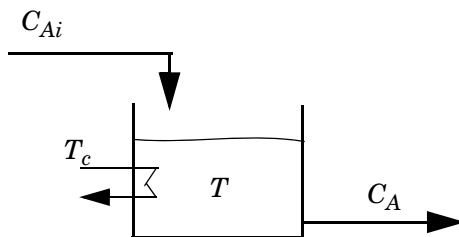
Figure 1-3: CSTR Schematic

From the Model Predictive Control Toolbox point for view, T and C_A would be plant outputs, and C_{Ai} and T_c would be inputs. More specifically, C_{Ai} would be an independent disturbance input, and T_c would be a manipulated variable (actuator).

There is one manipulated variable (the coolant temperature), so it's impossible to hold both T and C_A at setpoints. Controlling T would usually be a high priority. Thus, you might set the output weight for T much larger than that for C_A. In fact, you might set the C_A weight to zero, allowing C_A to float within an *acceptable operating region* (to be defined by constraints).

Move Suppression

If the controller focuses exclusively on setpoint tracking, it might choose to make large manipulated-variable adjustments. These could be impossible to

achieve. They could also accelerate equipment wear or lead to control system instability.

Thus, the Model Predictive Controller also monitors a weighted sum of controller adjustments, calculated according to the following equation:

$$S_{\Delta u}(k) = \sum_{i=1}^{M} \sum_{j=1}^{n_{mv}} \left\{ w_j^{\Delta u} \Delta u_j(k+i-1) \right\}^2$$

where M is the control horizon, n_{mv} is the number of manipulated variables, $\Delta u_j(k+i-1)$ is the predicted adjustment (i.e., *move*) in manipulated variable j at future (or current) sampling interval $k+i-1$, and $w_j^{\Delta u}$ is a weight, which must be zero or positive. Increasing $w_j^{\Delta u}$ forces the controller to make smaller, more cautious Δu_j moves. In many cases (but not all) this will have the following effects:

• The controller's setpoint tracking will degrade

• The controller will be less sensitive to prediction inaccuracies (i.e., more *robust*)

Setpoints on Manipulated Variables

In most applications, the controller's manipulated variables (MVs) should move freely (within a constrained region) to compensate for disturbances and stepoint changes. An attempt to hold an MV at a point within the region would degrade output setpoint tracking.

On the other hand, some plants have more MVs than output setpoints. In such a plant, if all manipulated variables were allowed to move freely, the MV values needed to achieve a particular setpoint or to reject a particular disturbance would be non-unique. Thus, the MVs would drift within the operating space.

A common approach is to define setpoints for "extra" MVs. These setpoints usually represent operating conditions that improve safety, economic return, etc. The Model Predictive Control Toolbox design includes an additional term to accommodate such cases, as follows:

$$S_u(k) = \sum_{i=1}^{M} \sum_{j=1}^{n_{mv}} \left\{ w_j^u [\bar{u}_j - u_j(k+i-1)] \right\}$$

where $\bar{u}_j$ is the manipulated variable setpoint (nominal value) for the jth MV, and w_j^u is the corresponding weight.

Constraints

Constraints may be either *hard* or *soft*. A hard constraint must not be violated. Unfortunately, under some conditions a constraint violation might be unavoidable (e.g., an unexpected, large disturbance), and a realistic controller must allow for this.

The Model Predictive Control Toolbox does so by *softening* each constraint, making a violation mathematically acceptable, though discouraged. The designer may specify the degree of softness in each case, making selected constraints less likely to be violated than others. See "Optimization Problem" on page 2-5 for the mathematical details.

Briefly, you specify a *tolerance band* for each constraint. If the tolerance band is zero, the constraint is hard (no violation allowed). Increasing the tolerance band softens the constraint.

The tolerance band *is not* a limit on the constraint violation, however. (If it were, you would still have a hard constraint.) You need to view it relative to other constraints.

For example, suppose you have two constraints, one on a temperature and the other on a flow rate. You specify a tolerance band of 2 degrees on the temperature constraint, and 20 kg/s on the flow rate constraint. The Model Predictive Controller assumes that violations of these magnitudes are of *equal concern*, and should be handled accordingly.

State Estimation

At the beginning of each sampling instant the controller estimates the current plant state. Accurate knowledge of the state improves prediction accuracy, which, in turn, improves controller performance.

If all plant states were measured, the state estimation problem would be relatively simple, requiring consideration of measurement noise effects only. Unfortunately, the internal workings of a typical plant are unmeasured, and the controller must estimate their current values from the available measurements. It also estimate the values of any sustained, unmeasured disturbances.

The Model Predictive Control Toolbox provides a default state estimation strategy, which the designer may customize. For details, see "State Estimation" on page 2-8.

Blocking

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

Figure 1-4 shows an alternative *blocked* strategy – again with 4 planned moves – in which the first 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 1-4 the block durations are 2, 2, 2, and 3. (Their sum must equal P.)
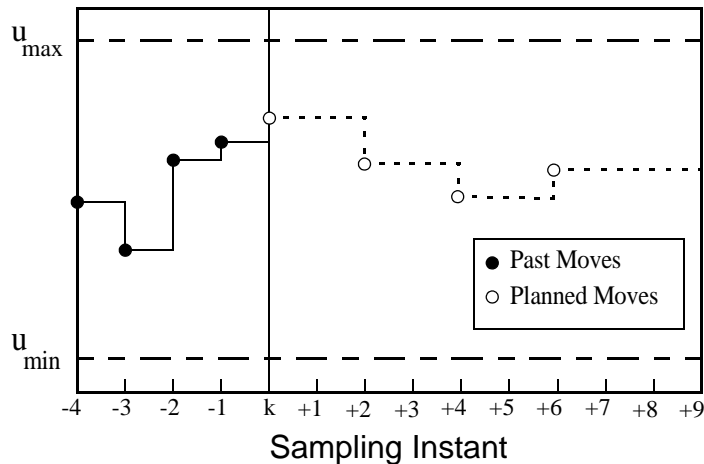
Figure 1-4: Blocking Example with 4 Moves

As for the default (unblocked) mode, only the current move, u_k, actually goes to the plant. Thus, as shown in Figure 1-4, the controller has made a plant adjustment at each sampling instant.

So why use blocking? When $P \gg M$ (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.

See the subsequent case study examples and the literature for more discussion and MIMO design guidelines.

2

MPC Problem Setup

Prediction Model

The linear model used in the Model Predicitive Control Toolbox for prediction and optimization is depicted in Figure 2-1.
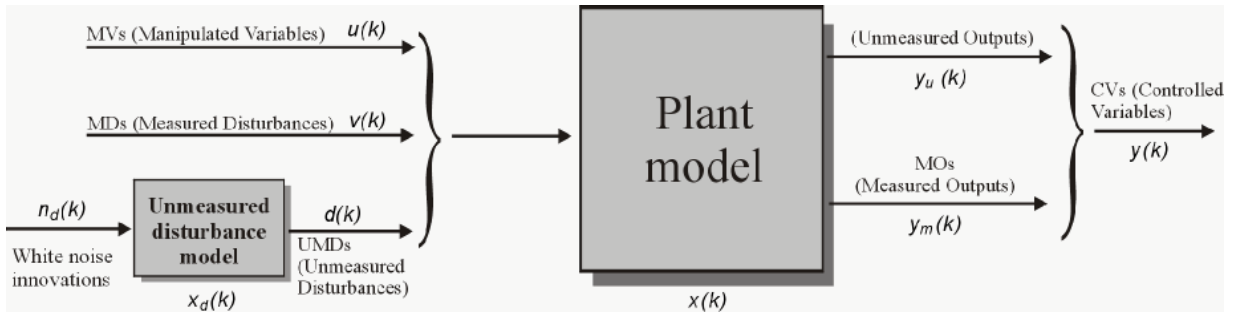
Figure 2-1: Model Used for Optimization

The model consists of

- A model of the *plant* to be controlled, whose inputs are the manipulated variables, the measured disturbances, and the unmeasured disturbances
- A model generating the unmeasured *disturbances*

Note When defining an MPC controller, you must specify a plant model. You do not need to specify a model generating the disturbances, as the controller setup assumes by default that unmeasured disturbances are generated by integrators driven by white noise (see "Output Disturbance Model" on page 2-9 and `setindist` on page 6-39).

The model of the plant is a linear time-invariant system described by the equations

$$x(k + 1) = Ax(k) + B_u u(k) + B_v v(k) + B_d d(k)$$
$$y_m(k) = C_m x(k) + D_{vm} v(k) + D_{dm} d(k)$$
$$y_u(k) = C_u x(k) + D_{vu} v(k) + D_{du} d(k) + D_{uu} u(k)$$

where $x(k)$ is the n_x-dimensional state vector of the plant, $u(k)$ is the n_u-dimensional vector of manipulated variables (MV), i.e., the command inputs, $v(k)$ is the n_v-dimensional vector of measured disturbances (MD), $d(k)$ is the n_d-dimensional vector of unmeasured disturbances (UD) entering the plant, $y_m(k)$ is the vector of measured outputs (MO), and $y_u(k)$ is the vector of unmeasured outputs (UO). The overall output vector $y(k)$ collects $y_m(k)$ and $y_u(k)$.

The Model Predictive Control Toolbox accepts both plant models specified as LTI objects, and models obtained from input/output data using the System Identification Toolbox (IDMODEL objects), see Using Identified Models (p. 2-19).

In the above equations $d(k)$ collects both state disturbances ($B_d \neq 0$) and output disturbances ($D_d \neq 0$).

Note A valid plant model for the Model Predicitive Control Toolbox cannot have direct feedthrough of manipulated variables $u(k)$ on the measured output vector $y_m(k)$.

The unmeasured disturbance $d(k)$ is modeled as the output of the linear time invariant system:

$$x_d(k+1) = \overline{A}x_d(k) + \overline{B}n_d(k) \tag{2-1}$$

$$d(k) = \overline{C}x_d(k) + \overline{D}n_d(k) \tag{2-2}$$

The system described by the above equations is driven by the random Gaussian noise $n_d(k)$, having zero mean and unit covariance matrix. For instance, a step-like unmeasured disturbance is modeled as the output of an integrator. Input disturbance models as in the equations above can be manipulated by using the methods `getindist` on page 6-14 and `setindist` on page 6-39.

Note If continuous-time models are supplied, they are internally sampled with the controller's sampling time.

Offsets

In many practical applications, the matrices A, B, C, D of the model representing the process to control are obtained by linearizing a nonlinear dynamical system, such as

$$x' = f(x, u, v, d)$$

$$y = h(x, u, v, d),$$

at some nominal value $x=x_0$, $u=u_0$, $v=v_0$, $d=d_0$. In these equations x' denotes either the time derivative (continuous time model) or the successor $x(k+1)$ (discrete time model). As an example, x_0, u_0, v_0, d_0 may be obtained by using `TRIM` on a simulink model describing the nonlinear dynamical equations, and A, B, C, D by using `LINMOD`. The linearized model has the form

$$x' \cong f(x_0, u_0, v_0, d_0) + \nabla_x f(x_0, u_0, v_0, d_0)(x - x_0) + \nabla_u f(x_0, u_0, v_0, d_0)(u - u_0) \\ + \nabla_v f(x_0, u_0, v_0, d_0)(v - v_0) + \nabla_d f(x_0, u_0, v_0, d_0)(d - d_0)$$

$$y \cong h(x_0, u_0, v_0, d_0) + \nabla_x h(x_0, u_0, v_0, d_0)(x - x_0) + \nabla_u h(x_0, u_0, v_0, d_0)(u - u_0) \\ + \nabla_v h(x_0, u_0, v_0, d_0)(v - v_0) + \nabla_d h(x_0, u_0, v_0, d_0)(d - d_0)$$

The matrices A, B, C, D of the model are readily obtained from the Jacobian matrices appearing in the equations above.

The linearized dynamics are affected by the constant terms $F=f(x_0, u_0, v_0, d_0)$ and $H=h(x_0, u_0, v_0, d_0)$. For this reason the MPC algorithm internally adds a measured disturbance $v=1$, so that F and H can be embedded into B_v and D_v, respectively, as additional columns.

Note Nonzero offset values d_0 for unmeasured disturbances, while relevant for obtaining the linearized model matrices, are not relevant for the MPC problem setup. In fact, only $d-d_0$ can be estimated from output measurements.

Optimization Problem

Assume that the estimates of $x(k)$, $x_d(k)$ are available at time k (for state estimation, see "State Estimation" on page 2-8). The MPC control action at time k is obtained by solving the optimization problem

(2-3)

$$\min_{\Delta u(k|k),\,\ldots,\,\Delta u(m-1+k|k),\,\varepsilon} \left\{ \sum_{i=0}^{p-1} \left(\sum_{j=1}^{n_y} \left| w_{i+1,j}^{y}(y_j(k+i+1|k) - r_j(k+i+1)) \right|^2 \right. \right.$$

$$\left. \left. + \sum_{j=1}^{n_u} \left| w_{i,j}^{\Delta u} \Delta u_j(k+i|k) \right|^2 + \sum_{j=1}^{n_u} \left| w_{i,j}^{u}(u_j(k+i|k) - u_{j\text{target}}(k+i)) \right|^2 \right) + \rho_\varepsilon \varepsilon^2 \right\}$$

where the subscript "$(\ )_j$" denotes the *j-th* component of a vector, "$(k+i|k)$" 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, subject to

$$u_{jmin}(i) - \varepsilon V_{j\ min}^{u}(i) \le u_j(k+i|k) \le u_{jmax}(i) + \varepsilon V_{j\ max}^{u}(i)$$

$$\Delta u_{jmin}(i) - \varepsilon V_{j\ min}^{\Delta u}(i) \le \Delta u_j(k+i|k) \le \Delta u_{jmax}(i) + \varepsilon V_{j\ max}^{\Delta u}(i)$$

$$y_{jmin}(i) - \varepsilon V_{j\ min}^{y}(i) \le y_j(k+i+1|k) \le y_{jmax}(i) + \varepsilon V_{j\ max}^{y}(i) \qquad i = 0,\,\ldots,\,p-1$$

$$\Delta u(k+h|k) = 0,\, h = m,\,\ldots,\,p-1$$

$$\varepsilon \ge 0$$

with respect to the sequence of input increments $\{\Delta u(k|k),\ldots,\Delta u(m\text{-}1+k|k)\}$ and to the slack variable ε, and by setting $u(k)=u(k\text{-}1)+\Delta u(k|k)^*$, where $\Delta u(k|k)^*$ is the first element of the optimal sequence.

Note Although only the measured output vector $y_m(k)$ is fed back to the MPC controller, $r(k)$ 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 $r(k+i+1)=r(k)$ in Equation 2-3. In model predictive control the exploitation of future references is referred to as *anticipative action (*or *look-ahead* or *preview*). 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 advance (e.g. is coming from a Simulink block) or $v(k+i)$ is obtained from the workspace. In the prediction, $d(k+i)$ is instead obtained by setting $n_d(k+i)=0$ in Figure 2-1 and Figure 2-2.

$w^{\Delta u}_{i,j}, w^u_{i,j}, w^y_{i,j}$, are nonnegative weights for the corresponding variable. The smaller w, the less important is the behavior of the corresponding variable to the overall performance index.

$u_{j,min}$, $u_{j,max}$, $\Delta u_{j,min}$, $\Delta u_{j,max}$, $y_{j,min}$, $y_{j,max}$ are lower/upper bounds on the corresponding variables. In Equation 2-4, the constraints on u, Δu, and y are relaxed by introducing the slack variable $\varepsilon \geq 0$. The weight ρ_ε on the slack variable ε penalizes the violation of the constraints. The larger ρ_ε with respect to input and output weights, the more the constraint violation is penalized. The Equal Concern for the Relaxation (ECR) vectors $V^u_{min}, V^u_{max}, V^{\Delta u}_{min}, V^{Du}_{max}, V^y_{min}, V^y_{max}$ have nonnegative entries which represent the concern for relaxing the corresponding constraint; the larger V, the *soft*er the constraint. $V=0$ means that the constraint is a *hard* one that cannot be violated. By default, all input constraints are hard ($V^u_{min}=V^u_{max}=V^{\Delta u}_{min}=V^{\Delta u}_{max}=0$) and all output constraints are soft ($V^y_{min}=V^y_{max}=1$). As hard output constraints may cause infeasibility of the optimization problem (for instance, because of unpredicted disturbances, model mismatch, or just because of numerical round off), a warning message is produced if V^y_{min}, V^y_{max} are smaller than a given small value and automatically adjusted at that value. By default,

$$\rho_\varepsilon = 10^5 \max\left\{ w^{\Delta u}_{i,j}, w^u_{i,j}, w^y_{i,j} \right\} \tag{2-4}$$

Note that also ECRs can be time varying.

Vector $u_{\text{target}}(k+i)$ is a setpoint for the input vector. One typically uses u_{target} if the number of inputs is greater than the number of outputs, as a sort of lower-priority setpoint.

As mentioned earlier, 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 $k+1$.

The algorithm implemented in the Model Predictive Control Toolbox uses different procedures depending on the presence of constraints. If all the bounds are infinite, then the slack variable ε is removed, and the problem in Equation 2-3 and Equation 2-4 is solved analytically. Otherwise a Quadratic Programming (QP) solver is used. The matrices associated with the quadratic optimization problem are described in "QP Matrices" on page 2-12.

Since output constraints are always soft, the QP problem is never infeasible. If for numerical reasons the QP problem becomes infeasible, the second sample from the previous optimal sequence is applied, i.e. $u(k)=u(k\text{-}1)+\Delta^{*}u(k\,|\,k\text{-}1)$.

Note For reasons of numerical robustness, for constrained MPC problems the default value $\Delta u_{\mathrm{j},min}$ for unbounded input rates is -10 and the maximum allowed lower bound is -1e5. The default value for unconstrained problems is minus infinity.

State Estimation

As the states $x(k)$, $x_d(k)$ are not directly measurable, predictions are obtained from a state estimator. In order to provide more flexibility, the estimator is based on the model depicted in Figure 2-2.
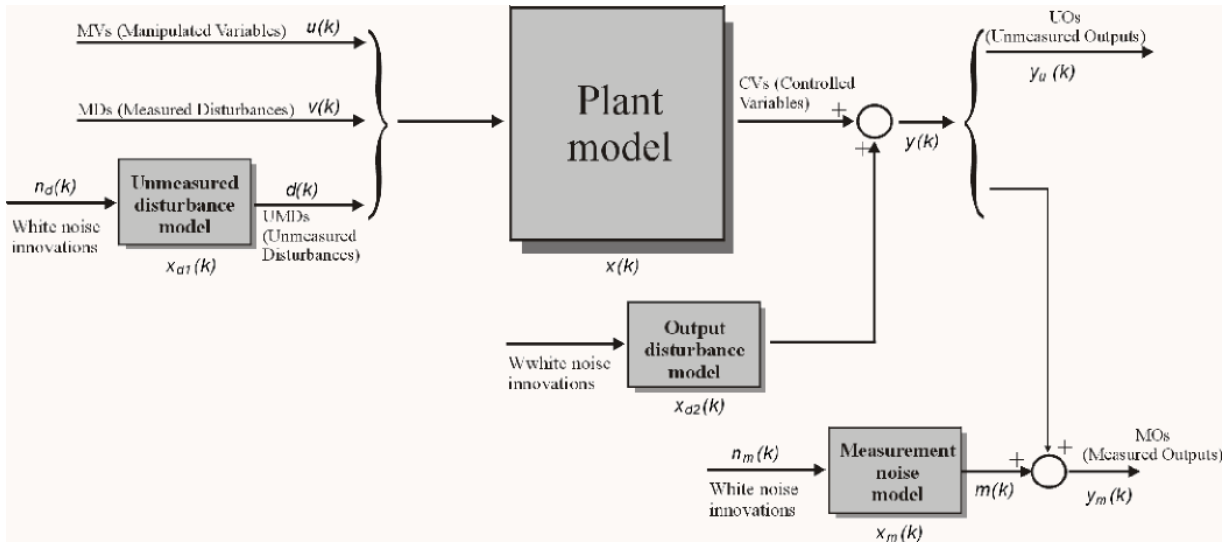
Figure 2-2: Model Used for State Estimation

Measurement Noise Model

We assume that the measured output vector $y_m(k)$ is corrupted by a measurement noise $m(k)$. The measurement noise $m(t)$ is the output of the linear time-invariant system

$$x_m(k + 1) = \tilde{A}x_m(k) + \tilde{B}n_m(k)$$

$$m(k) = \tilde{C}x_m(k) + \tilde{D}n_m(k).$$

The system described by these equations is driven by the random Gaussian noise $n_m(k)$, having zero mean and unit covariance matrix.

Note The objective of the MPC controller is to bring $y_u(k)$ and $[y_m(k)\text{-}m(k)]$ as close as possible to the reference vector $r(k)$. For this reason, the measurement noise model producing $m(k)$ is not needed in the prediction model used for optimization described in "Prediction Model" on page 2-2.

Output Disturbance Model

In order to guarantee asymptotic rejection of output disturbances, the overall model is augmented by an output disturbance model. By default, in order to reject constant disturbances due for instance to gain nonlinearities, the output disturbance model is a collection of integrators driven by white noise on measured outputs. Output integrators are added according to the following rule:

1 Measured outputs are ordered by decreasing output weight (in case of time-varying weights, the sum of the absolute values over time is considered for each output channel, and in case of equal output weight the order within the output vector is followed).

2 By following such order, an output integrator is added per measured outputs, unless there is a violation of observability or the user forces it (through the OutputVariables.Integrators property described in "OutputVariables" on page 8-5).

An arbitrary output disturbance model can be specified through the function setoutdist on page 6-44. See also setoutdist for ways to remove the default output integrators.

State Observer

The state observer is designed to provide estimates of $x(k)$, $x_d(k)$, $x_m(k)$, where $x(k)$ is the state of the plant model, $x_d(k)$ is the overall state of the input and output disturbance model, $x_m(k)$ is the state of the measurement noise model. The estimates are computed from the measured output $y_m(k)$ by the linear state observer

$$
\begin{bmatrix} \hat{x}(k|k) \\ \hat{x}_d(k|k) \\ \hat{x}_m(k|k) \end{bmatrix} = \begin{bmatrix} \hat{x}(k|k-1) \\ \hat{x}_d(k|k-1) \\ \hat{x}_m(k|k-1) \end{bmatrix} + M(y_m(k) - \hat{y}_m(k))
$$

$$
\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{x}_d(k+1|k) \\ \hat{x}_m(k+1|k) \end{bmatrix} = \begin{bmatrix} A\hat{x}(k|k) + B_u u(k) + B_v v(k) + B_d \overline{C} \hat{x}_d(k|k) \\ \overline{A}\hat{x}_d(k|k) \\ \tilde{A}\hat{x}_m(k|k) \end{bmatrix}
$$

$$
\hat{y}_m(k) = C_m \hat{x}(k|k-1) + D_{vm} v(k) + D_{dm}\overline{C}\hat{x}_d(k|k-1) + \tilde{C}\hat{x}_m(k|k-1)
$$

where m denotes the rows of C,D corresponding to measured outputs.

To prevent numerical difficulties in the absence of unmeasured disturbances, the gain M is designed using Kalman filtering techniques (see `kalman` in the Control System Toolbox) on the extended model

$$
\begin{bmatrix} x(k+1) \\ x_d(k+1) \\ x_m(k+1) \end{bmatrix} = \begin{bmatrix} A & B_d\overline{C} & 0 \\ 0 & \overline{A} & 0 \\ 0 & 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} x(k) \\ x_d(k) \\ x_m(k) \end{bmatrix} + \begin{bmatrix} B_u \\ 0 \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} B_v \\ 0 \\ 0 \end{bmatrix} v(k) + \begin{bmatrix} B_d\overline{D} & 0 & B_u & B_v \\ \overline{B} & 0 & 0 & 0 \\ 0 & \tilde{B} & 0 & 0 \end{bmatrix} \begin{bmatrix} n_d(k) \\ n_m(k) \\ n_u(k) \\ n_v(k) \end{bmatrix}
$$

$$
y_m(k) = \begin{bmatrix} C_m & D_{dm}\overline{C} & \tilde{C} \end{bmatrix} \begin{bmatrix} x(k) \\ x_d(k) \\ x_m(k) \end{bmatrix} + D_{vm}v(k) + \begin{bmatrix} \overline{D}_m & \tilde{D} & 0 & 0 \end{bmatrix} \begin{bmatrix} n_d(k) \\ n_m(k) \\ n_u(k) \\ n_v(k) \end{bmatrix} \quad \textbf{(2-5)}
$$

where $n_u(k)$ and $n_v(k)$ are additional unmeasured white noise disturbances having unit covariance matrix and zero mean, that are added on the vector of manipulated variables and the vector of measured disturbances, respectively, to ease the solvability of the Kalman filter design.

Note The overall state-space realization of the combination of plant and disturbance models must be observable for the state estimation design to succeed. The Model Predicitive Control Toolbox first checks for observability of the plant, provided that this is given in state-space form. After all models have been converted to discrete-time, delay-free, state-space form and combined together, observability of the overall extended model is checked (see `setestim` and "Construction and Initialization" on page 8-12).

Note also that observability is only checked numerically. Hence, for large models of badly conditioned system matrices, unobservability may be reported by the Model Predictive Control Toolbox even if the system is observable.

See also `getestim` on page 6-11 and `setestim` on page 6-36 for details on the methods that you can use to access and modify properties of the state estimator.

QP Matrices

This section describes the matrices associated with the MPC optimization problem described in "Optimization Problem" on page 2-5.

- "Prediction" on page 2-12
- "Optimization Variables" on page 2-13
- "Cost Function" on page 2-15
- "Constraints" on page 2-16

Prediction

Assume for simplicity that the disturbance model in Equation 2-1 and Equation 2-2 is a unit gain (i.e., $d(k)=n_d(k)$ is a white Gaussian noise). For simplicity, denote by

$$x \leftarrow \begin{bmatrix} x \\ x_d \end{bmatrix}, A \leftarrow \begin{bmatrix} A & B_d\overline{C} \\ 0 & \overline{A} \end{bmatrix}, B_u \leftarrow \begin{bmatrix} B_u \\ 0 \end{bmatrix}, B_v \leftarrow \begin{bmatrix} B_v \\ 0 \end{bmatrix}, B_d \leftarrow \begin{bmatrix} B_d\overline{D} \\ \overline{B} \end{bmatrix}, C \leftarrow \begin{bmatrix} C & D_d\overline{C} \end{bmatrix}$$

Then, the prediction model given by

$$x(k+1) = Ax(k) + B_u u(k) + B_v v(k) + B_d n_d(k)$$

$$y(k) = Cx(k) + D_v v(k) + D_d n_d(k).$$

Consider for simplicity the prediction of the future trajectories of the model performed at time $k=0$. We set $n_d(i)=0$ for all prediction instants i, and obtain

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

which gives

$$\begin{bmatrix} y(1) \\ \dots \\ y(p) \end{bmatrix} = S_x x(0) + S_{u1} u(-1) + S_u \begin{bmatrix} \Delta u(0) \\ \dots \\ \Delta u(p-1) \end{bmatrix} + H_v \begin{bmatrix} v(0) \\ \dots \\ v(p) \end{bmatrix}$$

where

$$S_x = \begin{bmatrix} CA \\ CA^2 \\ \ldots \\ CA^p \end{bmatrix} \in \Re^{pn_y \times n_x}, S_{u1} = \begin{bmatrix} CB_u \\ CB_u + CAB_u \\ \ldots \\ \sum_{h=0}^{p-1} CA^h B_u \end{bmatrix} \in \Re^{pn_y \times n_u}$$

$$S_u = \begin{bmatrix} CB_u & 0 & \ldots & 0 \\ CB_u + CAB_u & CB_u & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ \sum_{h=0}^{p-1} CA^h B_u & \sum_{h=0}^{p-2} CA^h B_u & \ldots & CB_u \end{bmatrix} \in \Re^{pn_y \times pn_u}$$

$$H_v = \begin{bmatrix} CB_v & D_v & 0 & \ldots & 0 \\ CAB_v & CB_v & D_v & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ CA^{p-1}B_v & CA^{p-2}B_v & CA^{p-3}B_v & \ldots & D_v \end{bmatrix} \in \Re^{pn_y \times (p+1)n_v}$$

Optimization Variables

Let m be the number of free control moves and denote by $z = [z_0; \ldots; z_{m-1}]$. Then,

$$\begin{bmatrix} \Delta u(0) \\ \ldots \\ \Delta u(p-1) \end{bmatrix} = J_M \begin{bmatrix} z_0 \\ \ldots \\ z_{m-1} \end{bmatrix} \tag{2-6}$$

where J_M depends on the choice of blocking moves. Together with the slack variable ε, vectors $z_0, \ldots, z_{m-1}$ constitute the free optimization variables of the optimization problem (in case of systems with a single manipulated variables, $z_0, \ldots, z_{m-1}$ are scalars).
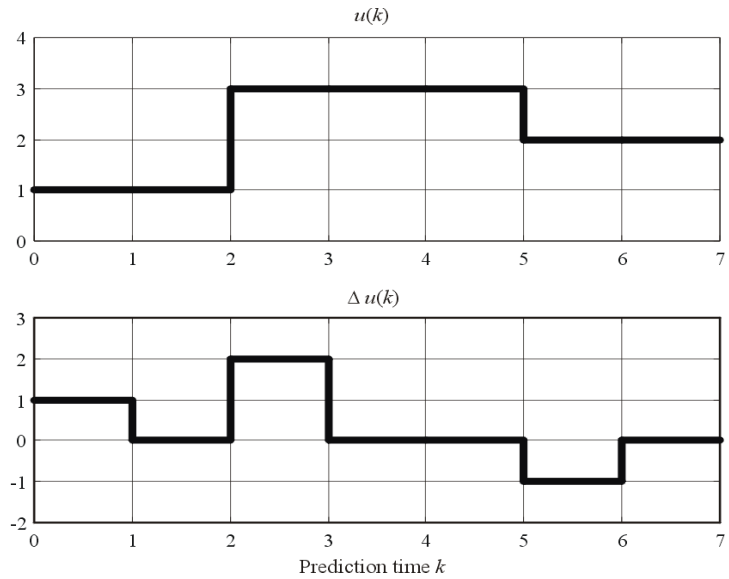
Figure 2-3: Blocking Moves: Inputs and Input Iincrements for `moves=[2 3 2]`

Consider for instance the blocking moves depicted in Figure 2-3, which corresponds to the choice `moves=[2 3 2]`, or, equivalently, $u(0)=u(1)$, $u(2)=u(3)=u(4)$, $u(5)=u(6)$, $\Delta u(0)=z_0$, $\Delta u(2)=z_1$, $\Delta u(5)=z_2$, $\Delta u(1)=\Delta u(3)=\Delta u(4)=\Delta u(6)=0$.

Then, the corresponding matrix J_M is

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

Cost Function

The function to be optimized is

$$
J(z, \varepsilon) = \left(\begin{bmatrix} u(0) \\ \dots \\ u(p-1) \end{bmatrix} - \begin{bmatrix} u_{\text{target}}(0) \\ \dots \\ u_{\text{target}}(p-1) \end{bmatrix} \right)^T W^2_u + \left(\begin{bmatrix} u(0) \\ \dots \\ u(p-1) \end{bmatrix} - \begin{bmatrix} u_{\text{target}}(0) \\ \dots \\ u_{\text{target}}(p-1) \end{bmatrix} \right)
$$

$$
+ \begin{bmatrix} \Delta u(0) \\ \dots \\ \Delta u(p-1) \end{bmatrix}^T W^2_{\Delta u} \begin{bmatrix} \Delta u(0) \\ \dots \\ \Delta u(p-1) \end{bmatrix}
$$

$$
+ \left(\begin{bmatrix} y(1) \\ \dots \\ y(1) \end{bmatrix} - \begin{bmatrix} r(1) \\ \dots \\ r(p) \end{bmatrix} \right)^T W^2_y \left(\begin{bmatrix} y(1) \\ \dots \\ y(1) \end{bmatrix} - \begin{bmatrix} r(1) \\ \dots \\ r(p) \end{bmatrix} \right) + \rho_\varepsilon \varepsilon^2
$$

where

$$
W_u = \text{diag}(w^u_{0,1}, w^u_{0,2}, \dots, w^u_{0,n_u}, \dots, w^u_{p-1,1}, w^u_{0p-1,2}, \dots, w^u_{p-1,n_u})
$$

$$
W_{\Delta u} = \text{diag}(w^{\Delta u}_{0,1}, w^{\Delta u}_{0,2}, \dots, w^{\Delta u}_{0,n_u}, \dots, w^{\Delta u}_{p-1,1}, w^{\Delta u}_{0p-1,2}, \dots, w^{\Delta u}_{p-1,n_u})
$$

$$
W_y = \text{diag}(w^y_{1,1}, w^y_{1,2}, \dots, w^y_{1,n_y}, \dots, w^y_{p,1}, w^y_{p,2}, \dots, w^y_{p,n_y})
$$

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

$$
J(z, \varepsilon) = \rho_\varepsilon \varepsilon^2 + z^T K_{\Delta u} z + 2 \left(\begin{bmatrix} r(1) \\ \dots \\ r(p) \end{bmatrix}^T K_r + \begin{bmatrix} v(0) \\ \dots \\ v(p) \end{bmatrix} K_v + u(-1)^T K_u \right. \tag{2-7}
$$

$$
\left. + \begin{bmatrix} u_{\text{target}}(0) \\ \dots \\ u_{\text{target}}(p-1) \end{bmatrix}^T K_{ut} + x(0)^T K_x \right) z + \text{constant}
$$

Note In order to keep the QP problem always strictly convex, if the condition number of the Hessian matrix $K_{\Delta U}$ is larger than 10^{12}, the quantity `10*sqrt(eps)` is added on each diagonal term. This may only occur when all input rates are not weighted ($W^{\Delta u}=0$) (see "Weights" on page 8-7).

Constraints

Let us now consider the limits on inputs, input increments, and outputs along with the constraint $\varepsilon \geq 0$.

$$
\begin{bmatrix}
y_{min}(1) - \varepsilon V^y{}_{min}(1) \\
\ldots \\
y_{min}(p) - \varepsilon V^y{}_{min}(p) \\
u_{min}(0) - \varepsilon V^u{}_{min}(0) \\
\ldots \\
u_{min}(p-1) - \varepsilon V^u{}_{min}(p-1) \\
\Delta u_{min}(0) - \varepsilon V^{\Delta u}{}_{min}(0) \\
\ldots \\
\Delta u_{min}(p-1) - \varepsilon V^{\Delta u}{}_{min}(p-1)
\end{bmatrix}
\leq
\begin{bmatrix}
y(1) \\
\ldots \\
y(p) \\
u(0) \\
\ldots \\
u(p-1) \\
\Delta u(0) \\
\ldots \\
\Delta u(p-1)
\end{bmatrix}
\leq
\begin{bmatrix}
y_{max}(1) + \varepsilon V^y{}_{max}(1) \\
\ldots \\
y_{max}(p) + \varepsilon V^y{}_{max}(p) \\
u_{max}(0) + \varepsilon V^u{}_{max}(0) \\
\ldots \\
u_{max}(p-1) + \varepsilon V^u{}_{max}(p-1) \\
\Delta u_{max}(0) + \varepsilon V^{\Delta u}{}_{max}(0) \\
\ldots \\
\Delta u_{max}(p-1) + \varepsilon V^{\Delta u}{}_{max}(p-1)
\end{bmatrix}
$$

Note Upper and lower bounds that are not finite are removed, as well as the input and input-increment bounds over blocked moves.

Similarly to what was done for the cost function, we can substitute $u(k)$, $\Delta u(k)$, $y(k)$, and obtain

$$M_z z + M_\varepsilon \varepsilon \le M_{\lim} + M_v \begin{bmatrix} v(0) \\ \dots \\ v(p) \end{bmatrix} + M_u u(-1) + M_x x(0) \qquad \textbf{(2-8)}$$

where matrices $M_z, M_\varepsilon, M_{\lim}, M_v, M_u, M_x$ are obtained from the upper and lower bounds and ECR values.

The matrices of the QP problem are built in function `mpc_buildmat`.

MPC Computation

This section describes how the MPC optimization problem is solved at each time step k (in mpcmove, mpc_sfun.dll, and mpcloop_engine.dll) by using the matrices built at initialization described in "QP Matrices" on page 2-12.

Unconstrained MPC

The optimal solution is computed analytically:

$$
z^* = -K^{-1}{}_{\Delta u}\left(\begin{bmatrix} r(1) \\ \dots \\ r(p) \end{bmatrix}^T K_r + \begin{bmatrix} v(0) \\ \dots \\ v(p) \end{bmatrix} K_v + u(-1)^T K_u + \begin{bmatrix} u_{\text{target}}(0) \\ \dots \\ u_{\text{target}}(p-1) \end{bmatrix}^T K_{ut} + x(0)^T K_x\right)^T
$$

and the MPC controller sets $\Delta u(k) = z^*{}_0$, $u(k) = u(k\text{-}1) + \Delta u(k)$.

Constrained MPC

The optimal solution z^*, ε^* is computed by solving the quadratic program described in Equation 2-7 and Equation 2-8, using the QP solver coded in the qpsolver.dll function (see qpdantz for more details).

Using Identified Models

The Model Predicitive Control Toolbox is able to handle plant models generated by the System Identification Toolbox from input/output measurements.

The Model Predicitive Control Toolbox labels control input signals as 'Manipulated', measured input disturbances as 'Measured', and unmeasured input disturbances as 'Unmeasured'. On the other hand, the System Identification Toolbox has a different naming rule, as it calls 'Measured' the inputs that are measurable quantities, and 'Noise' those that are not.

When you specify an identified model in the Model Predictive Control constructor as the plant model, the Model Predictive Control Toolbox treats 'Noise' signals as 'Unmeasured' input signals, and 'Measured' signals as 'Manipulated' signals, assuming that all measured inputs are also manipulated variables. You can later change later signal types, for instance to specify that some measured inputs are measured disturbances, rather than manipulated variables (see setname).

The Model Predicitive Control Toolbox internally converts the identified model you have provided as a plant model into the classical (A,B,C,D) state-space format. The columns of the B matrix originally related to 'Noise' channels are treated as the effect of unmeasured input disturbances on the state of the plant. On the other hand, the columns of the D matrix related to 'Noise' channels as treated as the effect of measurement noise superimposed on the output signal. Accordingly, the Model Predicitive Control Toolbox treats as the plant model the state-space model obtained from (A,B,C,D) by zeroing the columns of D related to 'Noise' channels. Those columns are instead used as a static noise model, or cascaded to an existing noise model if you have specified one. A unit static gain is assumed as the disturbance model, unless you have specified another one.

MPC Simulink Library

MPC Controller Block

Opening the Library

The MPC Simulink Library provides a single block representing the MPC controller.

The library can be opened from the main Simulink library or by typing mpclib from the command prompt.
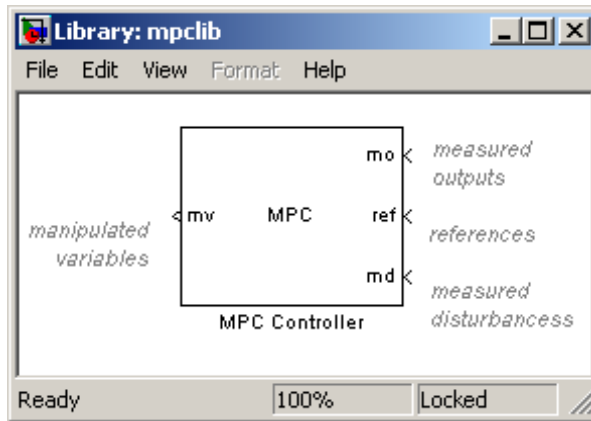
Figure 3-1: MPC Simulink Library

After copying the MPC Controller block in your diagram, you can double-click on the block and open the mask window.

MPC Controller Block Mask
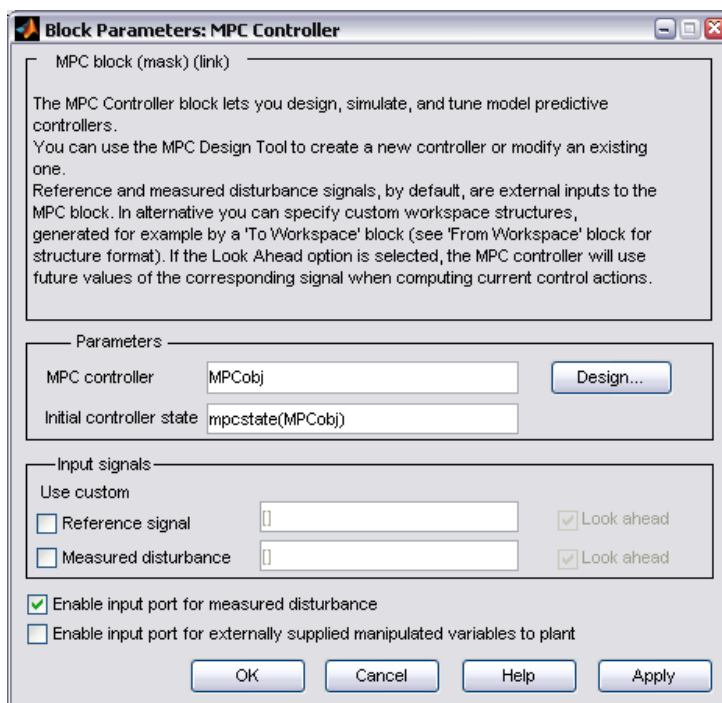
Figure 3-2: MPC Controller Block Mask

The mask requires that you specify a valid MPC controller object. There are three ways of providing an MPC controller object:

1 In the **MPC controller** edit box, enter the name of an MPC object that exists in the workspace

2 Enter the name of an MPC object that exists in the workspace and then click the **Design** button to open the MPC Design Tool and design the MPC controller object there.

3 Click the **Design** button without specifying an MPC controller object. The MPC Controller block prompts you to enter the number of manipulated varibales then will automatically to construct a default MPC controller by

3-3

linearizing the plant model from the Simulink diagram. This option requires Simulink Control Design. See "Importing a Plant Model" on page 5-9 for more information about creating linearized plant models with the MPC Toolbox. Refer to the Simulink Control Design documentation for more information about the linearization process.

Note Closed-loop simulations can be run while the MPC controller is edited in the MPC Design Tool. In this case, the controller parameters used for simulating the Simulink diagram are those currently specified in the MPC Design Tool, so that the parameters of the controller can be more easily tuned. Once the MPC Design Tool is closed, the designed controller must be exported as an MPC object to the workspace in order to be used in simulation.

Look Ahead and Signals from the Workspace

Reference and measured disturbance signals, by default, are taken from the Simulink diagram. They can be taken instead from the workspace. In this case, store the signal in a structure having the same format as used in the Simulink **From Workspace** and **To Workspace** blocks. This requires two fields: time and signals.

For example, to specify the sinusoidal reference signal $sin(t)$ over a time horizon of 10 seconds, assuming an MPC controller's sampling time Ts, use

```
time=(0:Ts:10);
ref.time=time;
ref.signals.values=sin(time);
```

An alternative would be to run a separate Simulink simulation to create the stored signal. Use the blocks required to define the signal (e.g., **Sine** in the above example), and store the result using a **To Workspace** block.

The **Look ahead** check box enables an anticipative action on the corresponding signal. It can only be enabled if the signal comes from the workspace.

See the demo mpcpreview for an illustrative example of enabling preview and reading signals from the workspace.

Initialization

The initial controller state must be a valid `mpcstate` object. See "MPC State Object" on page 8-13.

Using the MPC Toolbox with Real-Time Workshop

The C sources of the S-function executing the MPC Controller Block code are available in the `mpcutils/src` directory. You can build a real-time executable by pressing **Ctrl+B** on your Simulink diagram to invoke Real-Time Workshop® and build the model.

In some cases, it is necessary to copy the source files (`mpc_sfun.c`, `mpc_sfun.h`, `mpc_common.c`, `mat_macros.h`, `dantzgmp.h`, `dantzgmp_solver.c`) to a visible directory, such as the current directory `'.'`, or `'C:\MATLAB\rtw\c\src'`.

The MPC Controller block can be also used to produce real-time executables that run under xPC Target.

4

Case-Study Examples

Introduction

This chapter describes some typical MPC applications. Familiarity with LTI models (from the Control System Toolbox) and Simulink block diagrams will make the examples easier to understand, but you can skip the modeling details if you wish.

The first example designs a servomechanism controller. The specifications require a fast servo response despite constraints on a plant input and a plant output.

The second example controls a paper machine headbox. The process is nonlinear, and has three outputs, two manipulated inputs, and two disturbance inputs, one of which is measured for feedforward control.

The third example demonstrates the Model Predictive Control Toolbox bumpless transfer between manual and automatic operation.

Servomechanism Controller
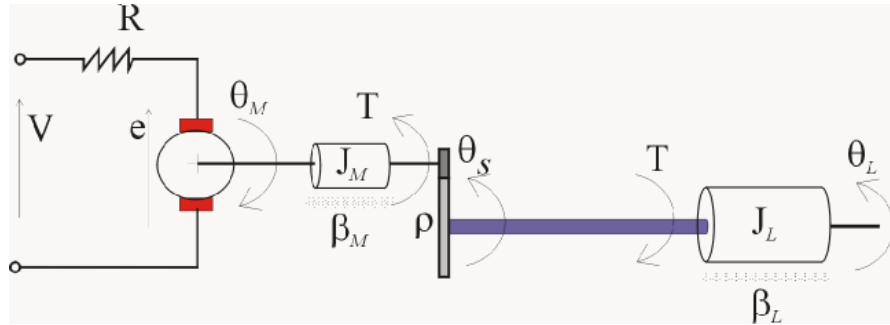
Figure 4-1: Position Servomechanism Schematic

System Model

A position servomechanism consists of a DC motor, gearbox, elastic shaft, and a load (see Figure 4-1). The differential equations representing this system are

$$\dot{\omega}_L = -\frac{k_\theta}{J_L}\Big(\theta_L - \frac{\theta_M}{\rho}\Big) - \frac{\beta_L}{J_L}\omega_L$$

$$\dot{\omega}_M = \frac{k_T}{J_M}\Big(\frac{V - k_T\omega_M}{R}\Big) - \frac{\beta_M\omega_M}{J_M} + \frac{k_\theta}{\rho J_M}\Big(\theta_L - \frac{\theta_M}{\rho}\Big)$$

where V is the applied voltage, T is the torque acting on the load, $\omega_L = \dot{\theta}_L$ is the load's angular velocity, $\omega_M = \dot{\theta}_M$ is the motor shaft's angular velocity, and the other symbols represent constant parameters (see Table 4-1 for more information on these).

If we define the state variables as $x_p = \begin{bmatrix} \theta_L & \omega_L & \theta_M & \omega_M \end{bmatrix}^T$, we can convert the above model to an LTI state-space form:

$$\dot{x}_p = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\dfrac{k_\theta}{J_L} & -\dfrac{\beta_L}{J_L} & \dfrac{k_\theta}{\rho J_L} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{k_\theta}{\rho J_M} & 0 & -\dfrac{k_\theta}{\rho^2 J_M} & -\dfrac{\beta_M + k_T^2/R}{J_M} \end{bmatrix} x_p + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dfrac{k_T}{R J_M} \end{bmatrix} V$$

$$\theta_L = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} x_p$$

$$T = \begin{bmatrix} k_\theta & 0 & -\dfrac{k_\theta}{\rho} & 0 \end{bmatrix} x_p$$

Table 4-1: Parameters Used in the Servomechanism Model

Symbol	Value (SI units)	Definition
k_θ	1280.2	Torsional rigidity
k_T	10	Motor constant
J_M	0.5	Motor inertia
J_L	$50 J_M$	Load inertia
ρ	20	Gear ratio
β_M	0.1	Motor viscous friction coefficient
β_L	25	Load viscous friction coefficient
R	20	Armature resistance

Control Objectives and Constraints

The controller must set the load's angular position, θ_L, at a desired value by adjusting the applied voltage, V. The only measurement available for feedback is θ_L.

The elastic shaft has a finite shear strength, so the torque, T, must stay within specified limits

$$|T| \leq 78.5 \ N \, m$$

Also, the applied voltage must stay within the range

$$|V| \leq 220 \ V$$

From an input/output viewpoint, the plant has a single input, V, which is manipulated by the controller. It has two outputs, one measured and fed back to the controller, θ_L, and one unmeasured, T.

Defining the Plant Model

The first step in a design is to define the plant model. The following commands are from the mpcdemos file mpcmotormodel.m, which you can run instead of entering the commands manually.

```
% DC-motor with elastic shaft
%
%Parameters (MKS)
%-----------------------------------------------------------
Lshaft=1.0;       %Shaft length
dshaft=0.02;      %Shaft diameter
shaftrho=7850;    %Shaft specific weight (Carbon steel)
G=81500*1e6;      %Modulus of rigidity
tauam=50*1e6;     %Shear strength
Mmotor=100;       %Rotor mass
Rmotor=.1;        %Rotor radius
Jmotor=.5*Mmotor*Rmotor^2; %Rotor axial moment of inertia
Bmotor=0.1;       %Rotor viscous friction coefficient (A CASO)
R=20;             %Resistance of armature
Kt=10;            %Motor constant
gear=20;          %Gear ratio
```

```
Jload=50*Jmotor; %Load inertia
Bload=25;        %Load viscous friction coefficient
Ip=pi/32*dshaft^4;              %Polar momentum of shaft
(circular) section
Kth=G*Ip/Lshaft;                %Torsional rigidity
(Torque/angle)
Vshaft=pi*(dshaft^2)/4*Lshaft;   %Shaft volume
Mshaft=shaftrho*Vshaft;          %Shaft mass
Jshaft=Mshaft*.5*(dshaft^2/4);   %Shaft moment of inertia
JM=Jmotor;
JL=Jload+Jshaft;
Vmax=tauam*pi*dshaft^3/16; %Maximum admissible torque
Vmin=-Vmax;

%Input/State/Output continuous time form
%----------------------------------------------------------
AA=[0            1           0                0;
    -Kth/JL      -Bload/JL   Kth/(gear*JL)    0;
    0            0           0                1;
    Kth/(JM*gear) 0          -Kth/(JM*gear^2)
-(Bmotor+Kt^2/R)/JM];

BB=[0;0;0;Kt/(R*JM)];
Hyd=[1 0 0 0];
Hvd=[Kth 0 -Kth/gear 0];
Dyd=0;
Dvd=0;

% Define the LTI state-space model
sys=ss(AA,BB,[Hyd;Hvd],[Dyd;Dvd]);
```

Controller Design Using MPCTOOL

The servomechanism model is linear, so you can use the Model Predictive
Control Toolbox design tool (mpctool) to configure a controller and test it.

Note To follow this example on your own system, first create the servomechanism model as explained in "Servomechanism Controller" on page 4-3. This defines the variable sys in your MATLAB workspace.

Opening MPCTOOL and Importing a Model

To begin, open the design tool by typing the following at the MATLAB prompt:

```
mpctool
```

Once the design tool has appeared, click the **Import Plant** button. The **Plant Model Importer** will appear (see Figure 4-2).

By default, the **Import from** option buttons are set to import from the MATLAB workspace, and the box at the upper right lists all LTI models defined there. In Figure 4-2, sys is the only available model, and it is selected. The **Properties** window lists the selected model's key attributes.
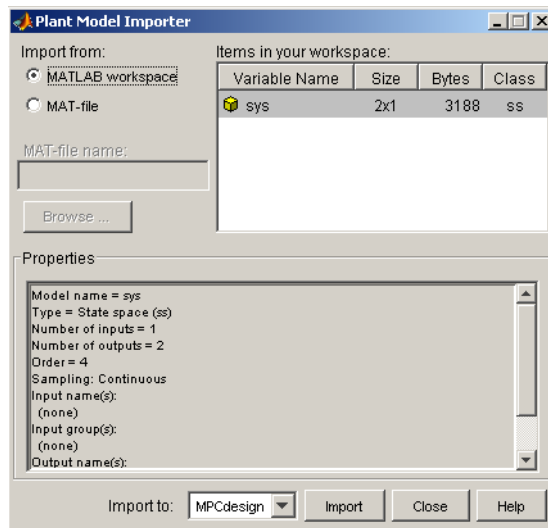
Figure 4-2: Import Dialog with the Servomechanism Model Selected

Make sure your servomechanism model, sys, is selected. Then click the **Import** button. You won't be importing more models, so close the import dialog window.

Meanwhile, the model has loaded, and tables now appear in the design tool's main window (see Figure 4-3). Note the diagram at the top, which ennumerates the model's input and output signals.
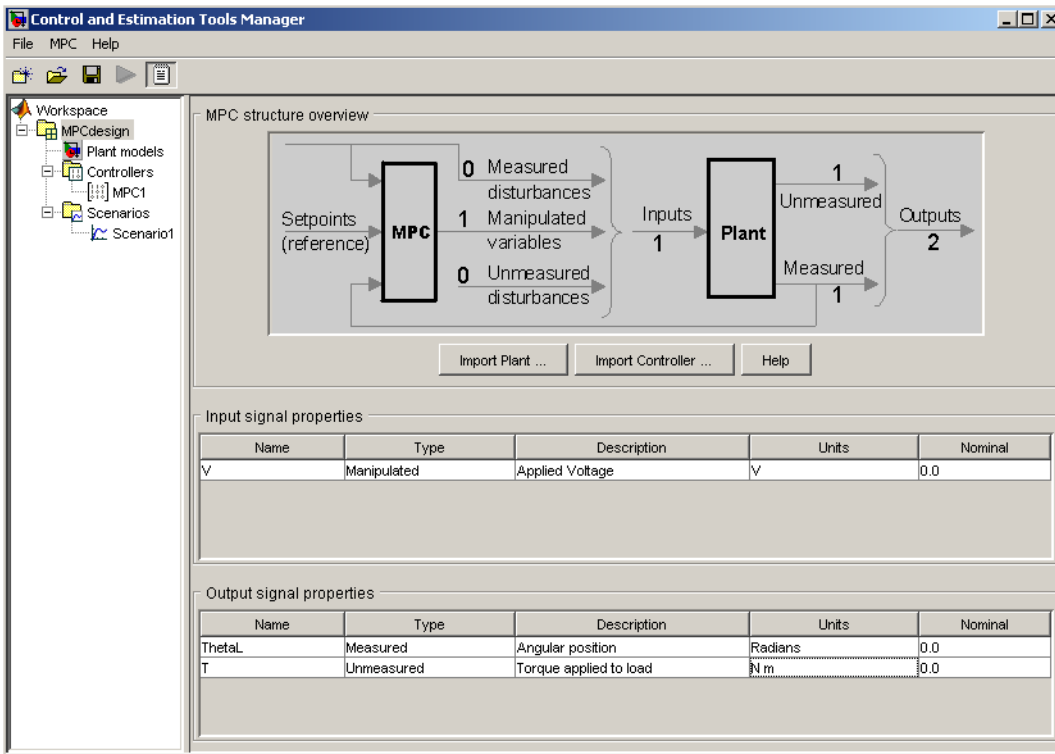
Figure 4-3: Design Tool After Importing the Plant Model and Specifying Signal Properties

Specifying Signal Properties

It's essential to specify *signal types* before going on. By default, the design tool assumes all plant inputs are manipulated, which is correct in this case. But it also assumes all outputs are measured, which is not. Specify that the second

output is unmeasured by clicking on the appropriate table cell and selecting the **Unmeasured** option.

You also have the option to change the default signal names (`In1`, `Out1`, `Out2`) to something more meaningful (e.g., `V`, `ThetaL`, `T`), enter descriptive information in the blank **Description** and **Units** columns, and specify a nominal initial value for each signal (the default is zero).

After you've entered all your changes, you should see a view similar to Figure 4-3. Notice that the upper graphic designates one output as measured, the other as unmeasured.

Navigation Using the Tree View

Now consider the design tool's left-hand frame. This *tree* is an ordered arrangement of *nodes*. Selecting (clicking) a node causes the corresponding view to appear in the right-hand frame. When the design tool starts, it creates a *root* node named **MPCdesign** and selects it, as in Figure 4-3.

The **Plant models** node is next in the hierarchy. Click on it to list the plant models being used in your design. (Each model name is editable.) The middle section displays the selected model's properties. There is also a space to enter notes describing the model's special features. Buttons allow you to import a new model or delete one you no longer need.

The next node is **Controllers**. You might see a + sign to its left, indicating that it contains subnodes. If so, click on the + sign to expand the tree (as shown in Figure 4-3). All the controllers in your design will appear here. By default, you have one: **MPC1**. In general, you might opt to design and test several alternatives.

Select **Controllers** to see a list of all controllers, similar to the **Plant models** view. The table columns show important controller settings: the plant model being used, the controller sampling period, and the prediction and control horizons. All are editable. For now, leave them at their default values.

The buttons on the **Controllers** view allow you to:

- **Import** a controller designed previously and stored either in your workspace or in a MAT-file
- **Export** the selected controller to your workspace
- Create a **New** controller, which will be initialized to the Model Predictive Control Toolbox defaults

> • **Copy** the selected controller to create a duplicate that you can modify
>
> • **Delete** the selected controller
>
> ### Specifying Controller Properties
>
> Select the **MPC1** subnode. The main pane should change to the controller design view shown in Figure 4-4.
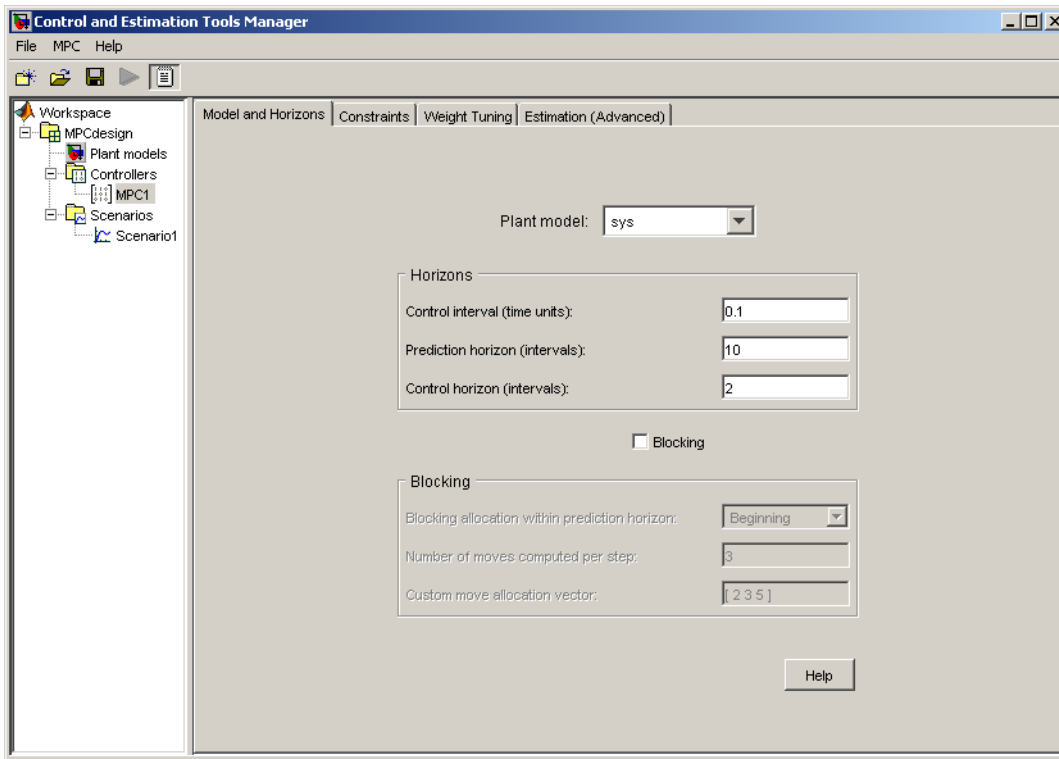
Figure 4-4: Controller Design View, Models and Horizons Pane

> If the selected **Prediction model** is continuous-time, as in this example, the **Control interval** (sampling period) defaults to 1. You need to change this to an application-appropriate value. Set it to 0.1 seconds (as shown in Figure 4-4). Leave the other values at their defaults for now.
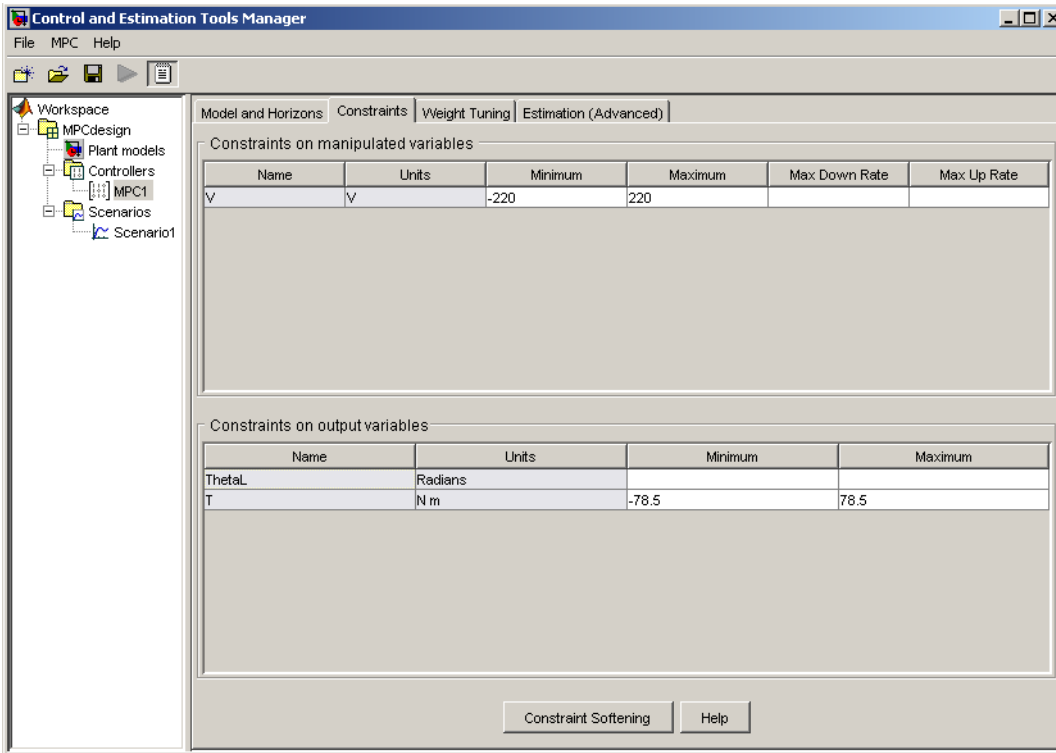
Figure 4-5: Controller Design View, Constraints Pane

Specifying Constraints

Next, select the **Constraints** tab. The view shown in Figure 4-5 appears. Enter the appropriate constraint values. Leaving a field blank implies that there is no constraint.

In general, it's good practice to include all known manipulated variable constraints, but it's unwise to enter constraints on outputs unless they are an essential aspect of your application. The limit on applied torque is such a constraint, as are the limits on applied voltage. The angular position has physical limits but the controller shouldn't attempt to enforce them, so you should leave the corresponding fields blank (see Figure 4-5).

The **Max down rate** should be nonpositive (or blank). It limits the amount a manipulated variable can decrease in a single control interval. Similarly, the **Max up rate** should be nonnegative. It limits the increasing rate. Leave both unconstrained (i.e., blank).

The shaded columns can't be edited. If you want to change this descriptive information, select the root node view and edit its tables. Such changes apply to all controllers in the design.

Weight Tuning

Next, select the **Weight Tuning** tab to obtain a view like that shown in Figure 4-6.

The *weights* specify trade-offs in the controller design. First consider the **Output weights**. The controller will try to minimize the deviation of each output from its *setpoint* or *reference* value. For each sampling instant in the prediction horizon, the controller multiplies predicted deviations for each output by the output's weight, squares the result, and sums over all sampling instants and all outputs. One of the controller's objectives is to minimize this sum, *i.e.*, to provide good *setpoint tracking* (See "Optimization Problem" on page 2-5 for more details.)

Here, the angular position should track its setpoint, but the applied torque can vary, provided that it stays within the specified constraints. Therefore, set the torque's weight to zero, which tells the controller that setpoint tracking is unnecessary for this output.

Similarly, it's acceptable for the applied voltage to deviate from nominal (it must in order to change the angular position!). Its weight should be zero (the default for manipulated variables). On the other hand, it's probably undesirable for the controller to make drastic changes in the applied voltage. The **Rate weight** penalizes such changes. Use the default, 0.1.

When setting the rates, the relative magnitudes are more important than the absolute values, and you must account for differences in the measurement scales of each variable. For example, if a deviation of 0.1 units in variable A is just as important as a deviation of 100 units in variable B, variable A's weight must be 1000 times larger than that for variable B.
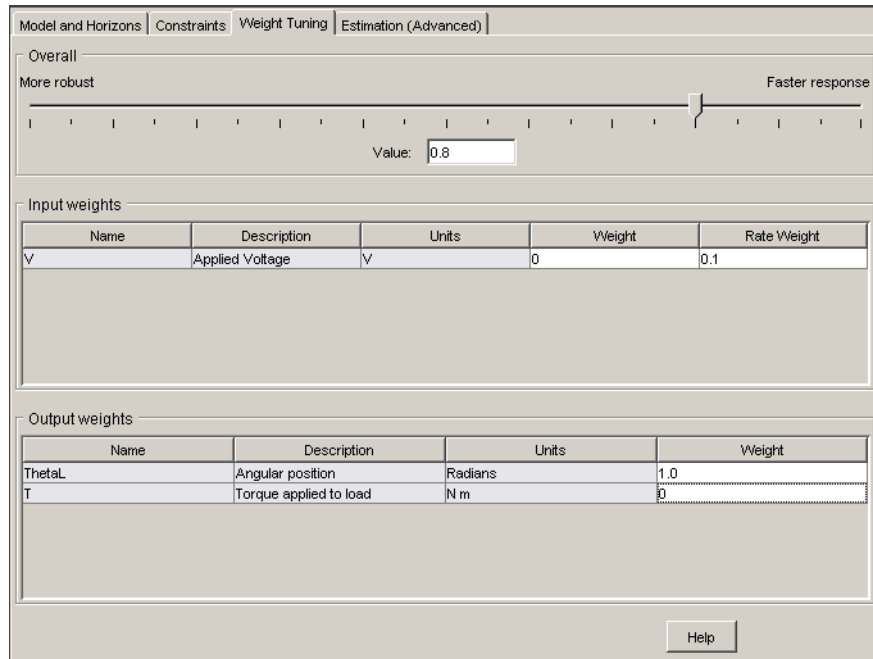
Figure 4-6: Controller Design View, Weight Tuning Pane

The tables allow you to weight individual variables. The slider at the top adjusts an overall trade-off between controller agressiveness and setpoint tracking. Moving the slider to the left places a larger overall penalty on manipulated variable changes, making them smaller. This usually increases controller robustness, but setpoint tracking becomes more sluggish.

The **Estimation** tab allows you to adjust the controller's response to unmeasured disturbances (not used in this example).
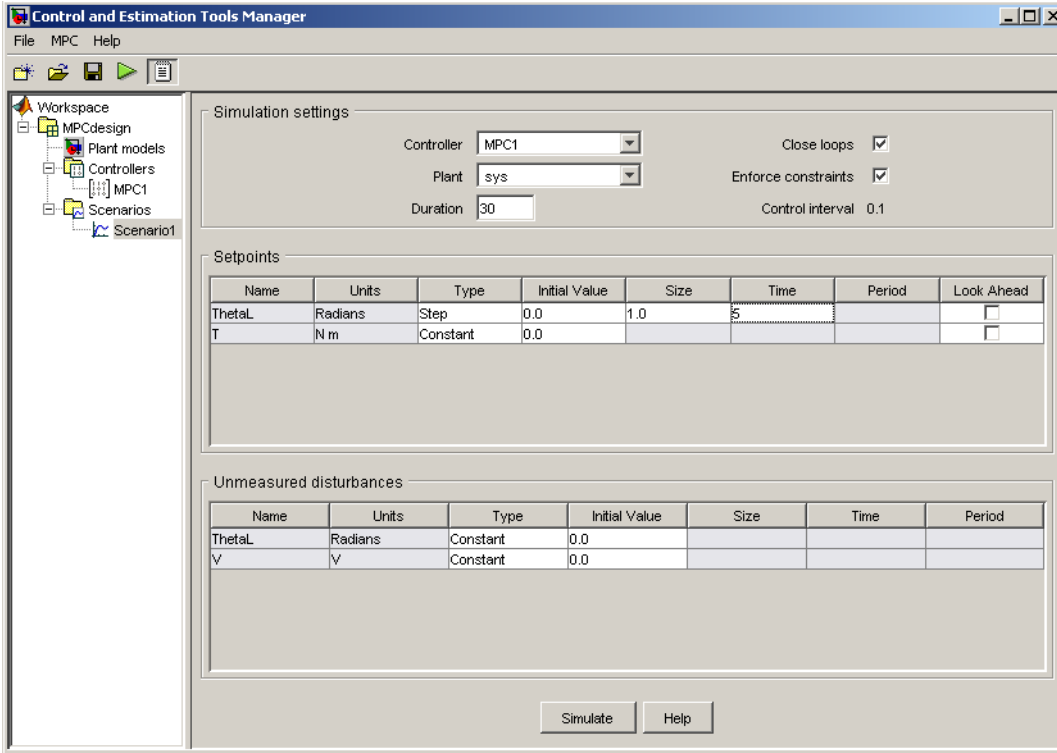
Figure 4-7: Simulation Settings View for "Scenario1"

Defining a Simulation Scenario

If you haven't already done so, expand the **Scenarios** node to show the **Scenario1** subnode (see Figure 4-3). Select **Scenario1** to obtain the view shown in Figure 4-7.

A *scenario* is a set of simulation conditions. As shown in Figure 4-7, you choose the controller to be used (from among controllers in your design), the model to act as the plant, and the simulation duration.

You must also specify all setpoints and disturbance inputs.

Duplicate the settings shown in Figure 4-7, which will test the controller's servo response to a unit-step change in the angular position setpoint. All other inputs are being held constant at their nominal values.

Note The **ThetaL** and **V** unmeasured disturbances allow you to simulate additive disturbances to these variables. By default, these disturbances are turned off, i.e., zero.

The **Look ahead** option designates that all future setpoint variations are known. In that case, the controller can adjust the manipulated variable(s) in advance to improve setpoint tracking. This would be unusual in practice, and is not being used here.
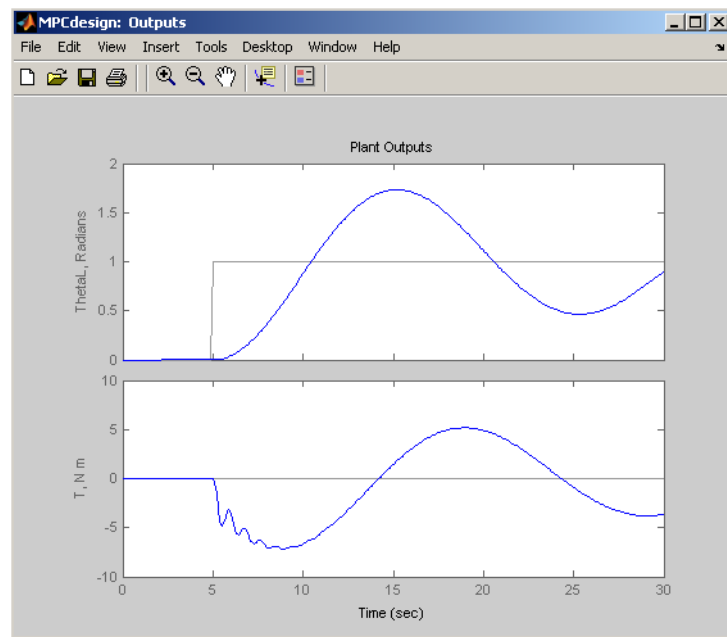
Figure 4-8: Response to Unit Step in the Angular Position Setpoint

Running a Simulation

Once you're ready to run the scenario, click the **Simulate** button or the green arrow on the toolbar.

Note The green arrow tool is available from any view once you've defined at least one scenario. It runs the *active scenario*, i.e., the one most recently selected or modified.

We obtain the results shown in Figure 4-8. The blue curves are the output signals, and the gray curves are the corresponding setpoints. The response is very sluggish, and hasn't settled within the 30-second simulation period.

Note The window shown in Figure 4-8 provides many of the customization features available in the Control System Toolbox `ltiview` and `sisotool` displays. Try clicking on a curve to obtain the numerical characteristics of the selected point, or right-clicking in the plot area to open a customization menu.

The corresponding applied voltage adjustments appear in a separate window (not shown) and are also very sluggish.

On the positive side, the applied torque stays well within bounds, as does the applied voltage.

Retuning to Achieve a Faster Servo Response

To obtain a more rapid servo response, navigate to the **MPC1 Weight Tuning** pane (select the **MPC1** node to get the controller design view, then select the **Weight Tuning** tab) and move the slider all the way to the right. Then click on the green arrow in the toolbar. Your results should now resemble Figure 4-9 and Figure 4-10.

The angular position now settles within 10 seconds following the step. The torque approaches its lower limit, but doesn't exceed it (see Figure 4-9) and the applied voltage stays within its limits (see Figure 4-10).
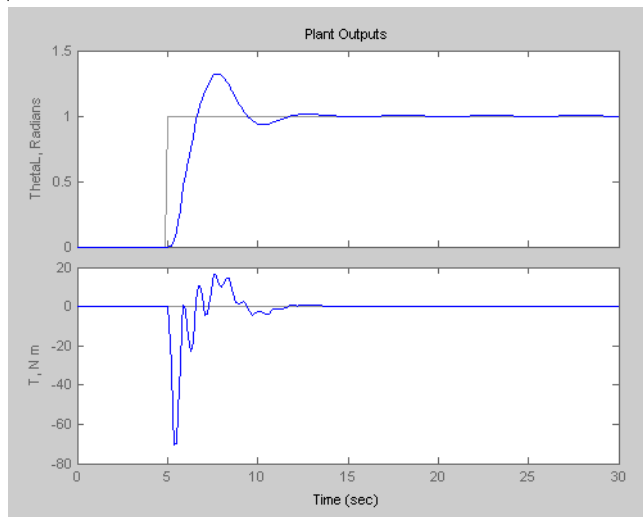
Figure 4-9: Faster Servo Response
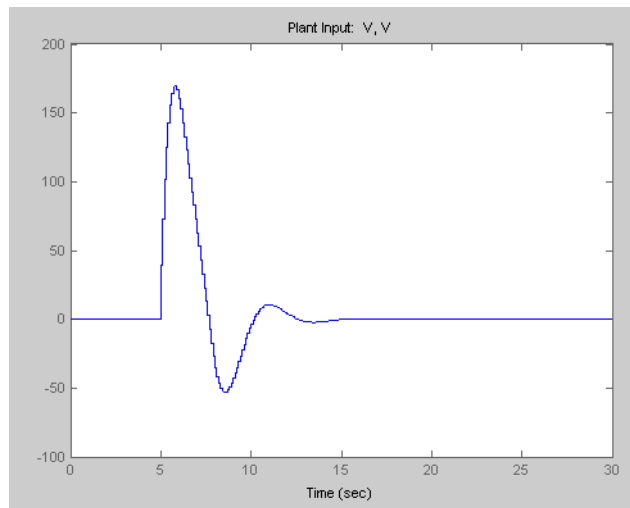
Figure 4-10: Manipulated Variable Adjustments Corresponding to Figure 4-9

4-17

Modifying the Scenario

Finally, increase the step size to π radians (select the **Scenario1** node and edit the tabular value).

As shown in Figure 4-11 and Figure 4-12, the servo response is essentially as good as before, and we avoid exceeding the torque constraint at -78.5 Nm, even though the applied voltage is saturated for about 2.5 seconds (see Figure 4-12).
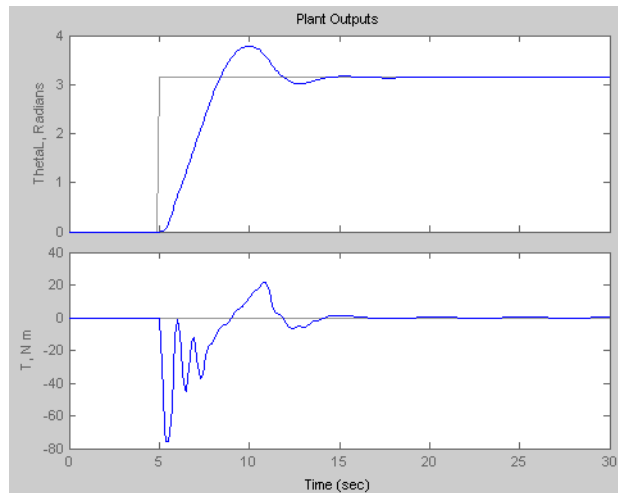
Figure 4-11: Servo Response for Step Increase of π Radians
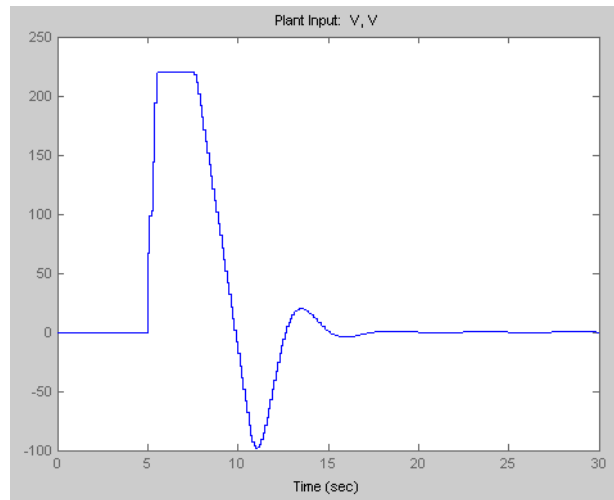
Figure 4-12: Voltage Adjustments Corresponding to Figure 4-11

Saving Your Work

Once you're satisfied with a controller's performance, you can export it to the workspace, for use in a Simulink block diagram or for analysis (or you can save it in a MAT-file).

To export a controller, right-click on its node and select **Export** from the resulting menu (or select the **Controllers** node, select the controller in the list, and click the **Export** button). A dialog like that shown in Figure 4-13 will appear.

The **Controller source** is the design from which you wish to extract a controller. There's only one in this example, but in general you might be working on several simultaneously. The **Controller to export** choice defaults to the controller most recently selected. Again, there's no choice in this case, but there could be in general. The **Name to assign** edit box allows you to rename the exported controller. (This will not change its name in the design tool.)

Figure 4-13: Exporting a Controller to the Workspace

Note When you exit the design tool you will be prompted to save the entire design in a MAT file. This allows you to reload it later using the **File/Load** menu option or the **Load** icon on the toolbar.

Using Model Predictive Control Toolbox Commands

Once you've become familiar with the toolbox, you may find it more convenient to build a controller and run a simulation using commands.

For example, suppose that you've already defined the model as discussed in "Defining the Plant Model" on page 4-5. Consider the following command sequence:

```
ManipulatedVariables = struct('Min', -220, 'Max', 220, 'Units',
'V');
OutputVariables(1) = struct('Min', -Inf, 'Max', Inf, 'Units',
'rad');
OutputVariables(2) = struct('Min', -78.5, 'Max', 78.5, 'Units',
'Nm');
Weights = struct('Input', 0, 'InputRate', 0.05, 'Output', [10 0]);
Model.Plant = sys;
Model.Plant.OutputGroup = {[1], 'Measured' ; [2], 'Unmeasured'};
Ts = 0.1;
PredictionHorizon = 10;
ControlHorizon = 2;
```

This creates several *structure* variables. For example, `ManipulatedVariables` defines the display units and constraints for the applied voltage (the manipulated plant input). `Weights` defines the tuning weights shown in Figure 4-6 (but the numerical values used here provide better performance). `Model` designates the plant model (stored in `sys`, which we defined earlier). The code also sets the `Model.Plant.OutputGroup` property to designate the second output as unmeasured.

Constructing an MPC Object

Use the mpc command to construct an MPC object called `ServoMPC`:

```
ServoMPC = mpc(Model, Ts, PredictionHorizon, ControlHorizon);
```

Like the LTI objects used to define linear, time-invariant dynamic models, an MPC object contains a complete definition of a controller.

Setting, Getting, and Displaying Object Properties

Once you've constructed an MPC object, you can change its properties as you would for other objects. For example, to change the prediction horizon, you could use one of the following commands:

```
ServoMPC.PredictionHorizon = 12;
```

or

```
set(ServoMPC, 'PredictionHorizon', 12);
```

For a listing of all the object's properties, you could type

```
get(ServoMPC)
```

To access a particular property (e.g., the control horizon), you could type

```
M = get(ServoMPC, 'ControlHorizon');
```

or

```
M = ServoMPC.ControlHorizon;
```

You can also set multiple properties simultaneously.

Set the following properties before continuing with this example:

```
set(ServoMPC, 'Weights', Weights, ...
    'ManipulatedVariables', ManipulatedVariables, ...
```

```
         'OutputVariables', OutputVariables);
```

Typing the name of an object without a terminating semicolon generates a formatted display of the object's properties. You can achieve the same effect using the display command:

```
display(ServoMPC)
```

Running a Simulation

The sim command performs a linear simulation. For example, the following code sequence defines constant setpoints for the two outputs, then runs a simulation.

```
TimeSteps = round(10/Ts);
r = [pi 0];
[y, t, u] = sim(ServoMPC, TimeSteps, r);
```

By default, the model used to design the controller (stored in ServoMPC) also represents the plant.

The sim command saves the output and manipulated variable sequences in variables y and u. For example,

```
subplot(311)
plot(t, y(:,1), [0 t(end)], pi*[1 1])
title('Angular Position (radians)');
subplot(312)
plot(t, y(:,2), [0 t(end)], [-78.5 -78.5])
title('Torque (nM)')
subplot(313)
stairs(t, u)
title('Applied Voltage (volts)')
xlabel('Elapsed Time (seconds)')
```

produces the custom plot shown in Figure 4-14. The plot includes the angular position's setpoint. The servo response settles within 5 seconds with no overshoot. It also displays the torque's lower bound, which becomes active after about 0.9 seconds but isn't exceeded. The applied voltage saturates between about 0.5 and 2.8 seconds, but the controller performs well despite this.
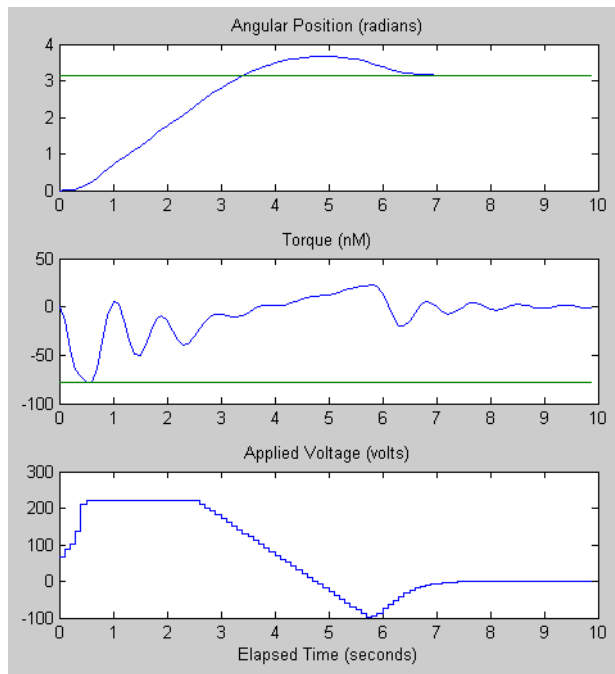
**Figure 4-14: Plotting the Output of the Sim Command**

Using MPC Tools in Simulink

Figure 4-15 is a Simulink block diagram for the servomechanism example. Most of the blocks are from the standard Simulink library. There are two exceptions:

- Servomechanism Model is an LTI System block from the Control System Toolbox library. The LTI model `sys` (which must exist in the workspace) defines its dynamic behavior. To review how to create this model, see "Defining the Plant Model" on page 4-5.

- MPC Controller is from the MPC Blocks library. Figure 4-16 shows the dialog box obtained by double-clicking on this block. You need to supply an MPC object, and `ServoMPC` is being used here. It must be in the Workspace before you run a simulation. The **Design** button opens the design tool, which allows you to create or modify the object. To review how to use commands to create `ServoMPC`, see "Constructing an MPC Object" on page 4-21.
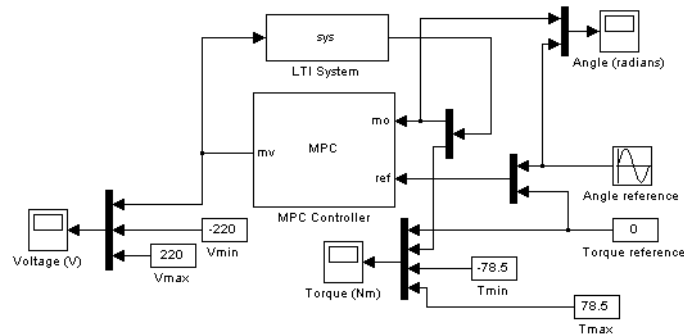
**Figure 4-15: Block Diagram for the Servomechanism Example**
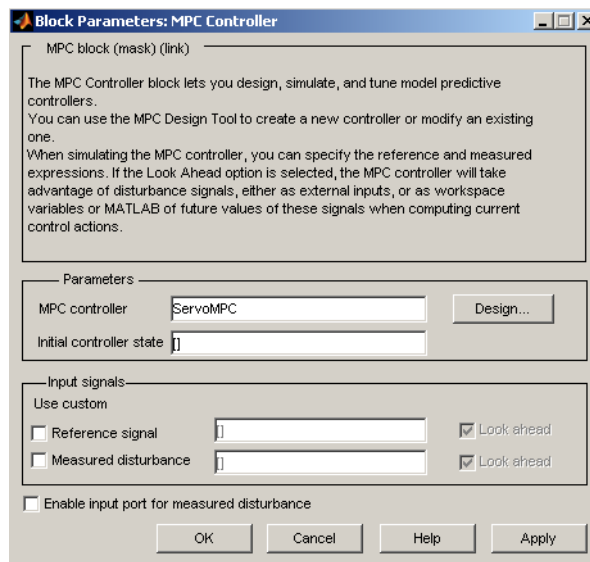
Figure 4-16: Model Predictive Control Toolbox Simulink Block Dialog

The key features of the diagram are as follows:

- The MPC Controller output is the plant input. The Voltage Scope block plots it (yellow curve). Minimum and maximum voltage values are shown as magenta and cyan curves.

- The plant output is a vector signal. The first element is the measured angular position. The second is the unmeasured torque. A Demux block separates them. The angular position feeds back to the controller and plots on the Angle scope (yellow curve). The torque plots on the Torque scope (with its lower and upper bounds).

- The position setpoint varies sinusoidally with amplitude π radians and frequency 0.4 rad/s. It also appears on the Angle scope (magenta curve).

Figure 4-17 shows the scope displays for a 20-second simulation. The angular position tracks the sinusoidal setpoint variations well despite saturation of the applied voltage. The setpoint variations are more gradual than the step changes used previously, so the torque stays well within its bounds.
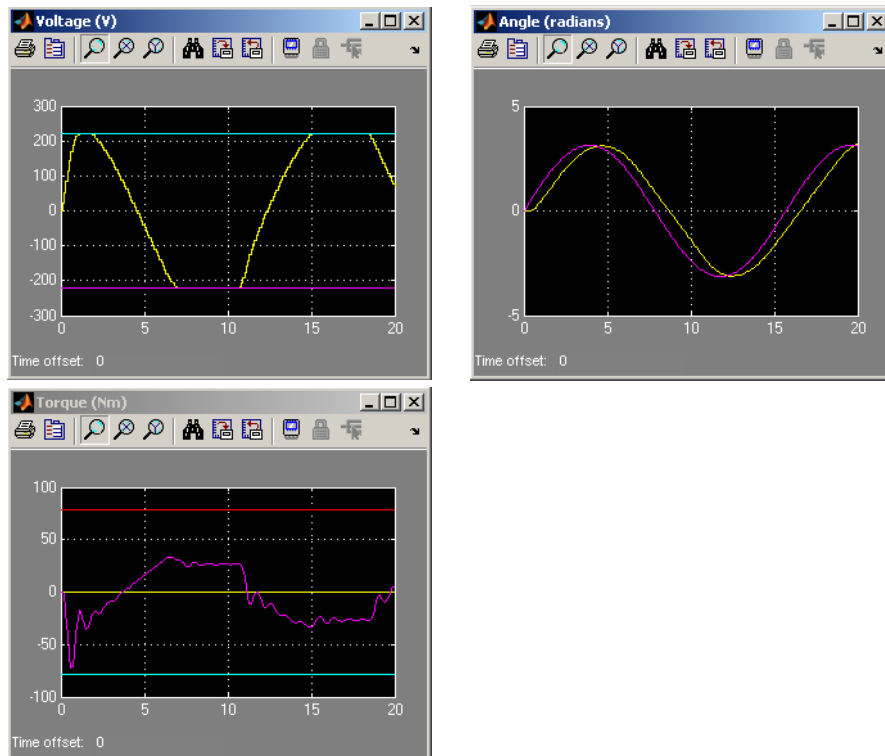
Figure 4-17: Servomechanism Simulation Scopes

Paper Machine Process Control
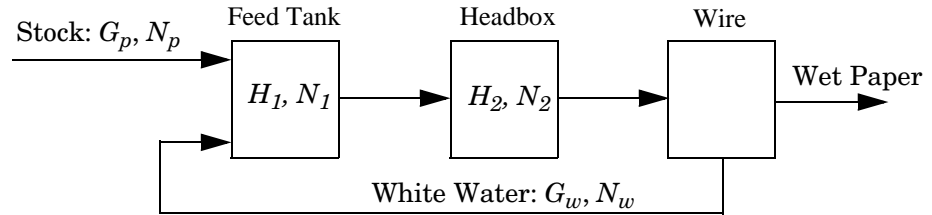
Figure 4-18: Schematic of Paper Machine Headbox Elements

Ying *et al.* [1] studied the control of consistency (percentage pulp fibers in aqueous suspension) and liquid level in a paper machine headbox, a schematic of which is shown in Figure 4-18. The process model is a set of ordinary differential equations (ODEs) in bilinear form. The states are

$$x = \begin{bmatrix} H_1 \ H_2 \ N_1 \ N_2 \end{bmatrix}^T$$

where H_1 is the liquid level in the feed tank, H_2 is the headbox liquid level, N_1 is the feed tank consistency, and N_2 is the headbox consistency. The measured outputs are

$$y = \begin{bmatrix} H_2 \ N_1 \ N_2 \end{bmatrix}^T$$

The primary control objectives are to hold H_2 and N_2 at setpoints. There are two manipulated variables

$$u = \begin{bmatrix} G_p \ G_w \end{bmatrix}^T$$

where G_p is the flow rate of stock entering the feed tank, and G_w is the recycled *white water* flow rate. The consistency of stock entering the feed tank, N_p, is a measured disturbance.

$$v = N_p$$

4-27

The white water consistency is an unmeasured disturbance.

$$d = N_w$$

Variables are normalized. All are zero at the nominal steady state and have comparable numerical ranges. Time units are minutes. The process is open-loop stable.

The mpcdemos folder contains file mpc_pmmodel.m, which implements the nonlinear model equations as a Simulink S-function. The input sequence is G_p, G_w, N_p, N_w, and the output sequence is H_2, N_1, N_2.

Linearizing the Nonlinear Model

The paper machine headbox model is easy to linearize analytically, yielding the following state space matrices:

```
A = [-1.9300          0          0          0
      0.3940    -0.4260          0          0
          0          0    -0.6300          0
      0.8200    -0.7840     0.4130    -0.4260];
B = [1.2740     1.2740          0          0
          0          0          0          0
      1.3400    -0.6500     0.2030     0.4060
          0          0          0          0];
C = [0     1.0000          0          0
          0          0     1.0000          0
          0          0          0     1.0000];
D = zeros(3,4);
```

Use these to create a continuous-time LTI state-space model, as follows:

```
PaperMach = ss(A, B, C, D);
PaperMach.InputName = {'G_p', 'G_w', 'N_p', 'N_w'};
PaperMach.OutputName = {'H_2', 'N_1', 'N_2'};
```

(The last two commands are optional; they improve plot labeling.)

As a quick check of model validity, plot its step responses as follows:

```
step(PaperMach);
```

The results appear in Figure 4-19. Note the following:

- The two manipulated variables affect all three outputs.
- They have nearly identical effects on H_2.
- The $G_w \to N_2$ pairing exhibits an inverse response.

These features make it difficult to achieve accurate, independent control of H_2 and N_2.
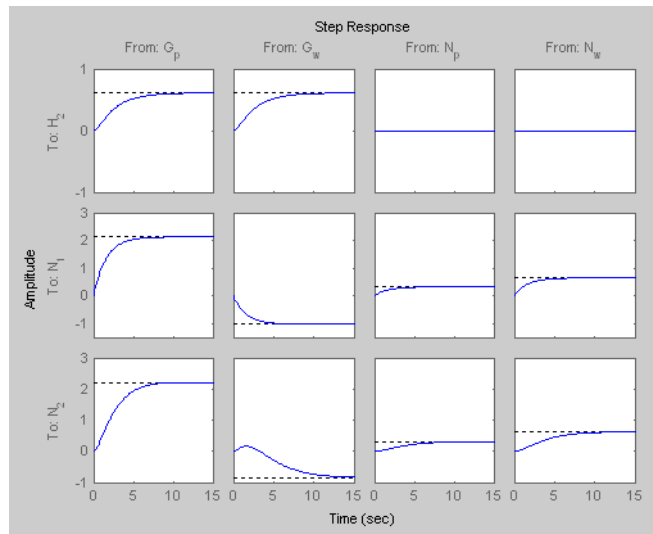
Figure 4-19: Linearized Paper Machine Model's Step Responses

MPC Design

Type

```
mpctool
```

to open the mpc design tool. Import your LTI PaperMach model as described in "Opening MPCTOOL and Importing a Model" on page 4-7.

Next, define signal properties, being sure to designate N_p and N_w as measured and unmeasured disturbances, respectively. Your specifications should resemble Figure 4-20.

Input signal properties

Name	Type	Description	Units	Nominal
G_p	Manipulated	Feed flow rate	kg/h	0.0
G_w	Manipulated	White water flow rate	kg/h	0.0
N_p	Meas. disturb.	Feed consistency	%	0.0
N_w	Unmeas. disturb.	White water consistency	%	0.0

Output signal properties

Name	Type	Description	Units	Nominal
H_2	Measured	Headbox level	m	0.0
N_1	Measured	Feed tank consistency	%	0.0
N_2	Measured	Head box consistency	%	0.0

Figure 4-20: Signal Properties for the Paper Machine Application

Initial Controller Design

If necessary, review "Specifying Controller Properties" on page 4-10. Then click the **MPC1** node and specify the following controller parameters (leaving others at their default values):

- **Models and Horizons**. Control interval = 2 minutes
- **Constraints**. For both G_p and G_w, Minimum = -10, Maximum = 10, Max down rate = -2, Max up rate = 2.
- **Weight Tuning**. For both G_p and G_w, Weight = 0, Rate weight = 0.4. For N_1, Weight = 0. (Other outputs have Weight = 1.)
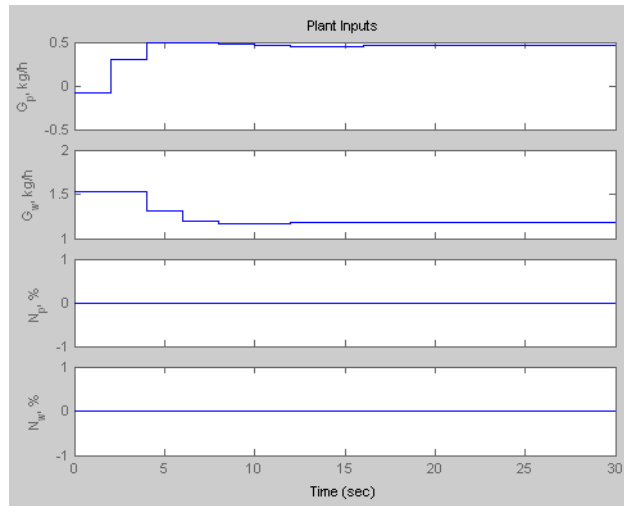
Servo Response

Finally, select the **Scenario1** node and define a servo-response test:

- Duration = 30
- H_2 setpoint = 1 (constant)

Simulate the scenario. You should obtain results like those shown in Figure 4-21 and Figure 4-22.
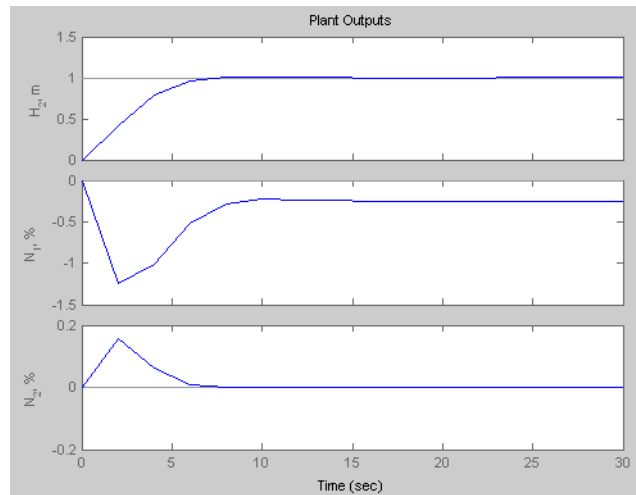
Figure 4-21: Servo Response for Unit Step in Headbox Level Setpoint

4-31

Figure 4-22: Manipulated Variable Moves Corresponding to Figure 4-21

Weight Tuning

The response time is about 8 minutes. We could reduce this by decreasing the control interval, reducing the manipulated variable rate weights, and/or eliminating the up/down rate constraints. The present design uses a conservative control effort, which would usually improve robustness, so we will continue with the current settings.

Note the steady-state error in N_1 (it's about -0.25 units in Figure 4-21). There are only two manipulated variables, so it's impossible to hold three outputs at setpoints. We don't have a setpoint for N_1 so we have set its weight to zero (see controller settings in "Initial Controller Design" on page 4-30). Otherwise, all three outputs would have exhibited steady-state error (try it).

Consistency control is more important than level control. Try decreasing the H_2 weight from 1 to 0.2. You should find that the peak error in N_2 decreases by almost an order of magnitude, but the H_2 response time increases from 8 to about 18 minutes (not shown). Use these modified output weights in subsequent tests.

Feedforward Control

To configure a test of the controller's feedforward response, define a new scenario by clicking on the **Scenarios** node, clicking the **New** button, and renaming the new scenario **Feedforward** (by editing its name in the tree or the summary list).

In the **Feedforward** scenario, define a step change in the measured disturbance, N_p, with **Initial value** = 0, **Size** = 1, **Time** = 10. All output setpoints should be zero. Set the **Duration** to 30 time units.

If response plots from the above servo response tests are still open, close them. Simulate the **Feedforward** scenario. You should find that the H_2 and N_2 outputs deviate very little from their setpoints (not shown).

Experiment with the "look ahead" feature. First, observe that in the simulation just completed the manipulated variables didn't begin to move until the disturbance occurred at t = 10 minutes. Return to the **Feedforward** scenario, select the **Look ahead** option for the measured disturbance, and repeat the simulation.

Notice that the manipulated variables begin changing *in advance* of the disturbance. This happens because the look ahead option uses known future values of the disturbance when computing its control action. For example, at time t = 0 the controller is using a prediction horizon of 10 control intervals (20 time units), so it "sees" the impending disturbance at t = 10 and begins to prepare for it. The output setpoint tracking improves slightly, but it was already so good that the improvement is insignificant. Also, it's unlikely that there would be advanced knowledge of a consistency disturbance, so clear the **Look ahead** check box for subsequent simulations.

Unmeasured Input Disturbance

To test the response to unmeasured disturbances, define another new scenario called **Feedback**. Configure it as for **Feedforward**, but set the measured disturbance, N_p, to zero (constant), and the unmeasured disturbance, N_w, to 1.0 (constant). This simulates a sudden, sustained, unmeasured disturbance occurring at time zero.

Running the simulation should yield results like those in Figure 4-23. The two controlled outputs (H_2 and N_2) exhibit relatively small deviations from their setpoints (which are zero). The settling time is longer than for the servo response (compare to Figure 4-21) which is typical.
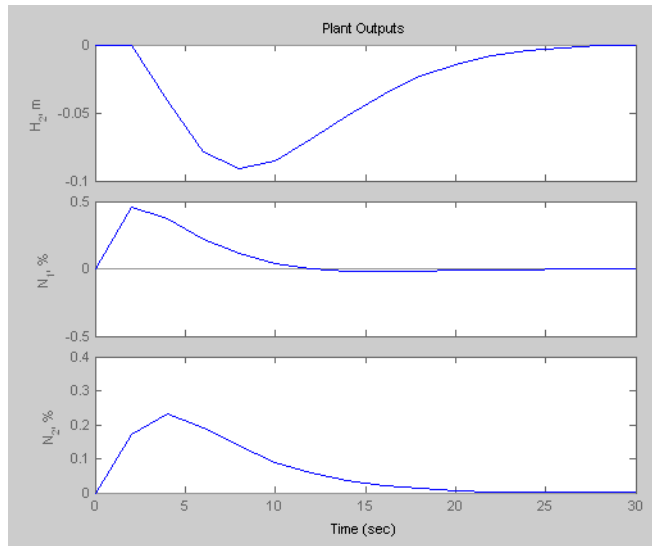
Figure 4-23: Feedback Scenario: Unmeasured Disturbance Rejection

One factor limiting performance is the chosen control interval of 2 time units. The controller can't respond to the disturbance until it first appears in the outputs, i.e., at $t = 2$. If you wish, experiment with larger and smaller intervals (modify the specification on the controller's **Model and Horizons** tab).

Effect of Estimator Assumptions

Another factor influencing the response to unmeasured disturbances (and model prediction error) is the estimator configuration. The results shown in Figure 4-23 are for the default configuration.

To view the default assumptions, select the controller node (**MPC1**), and select its **Estimation** tab. The resulting view should be as shown in Figure 4-24. The status message (bottom of figure) indicates that the Model Predictive Control Toolbox default assumptions are being used.
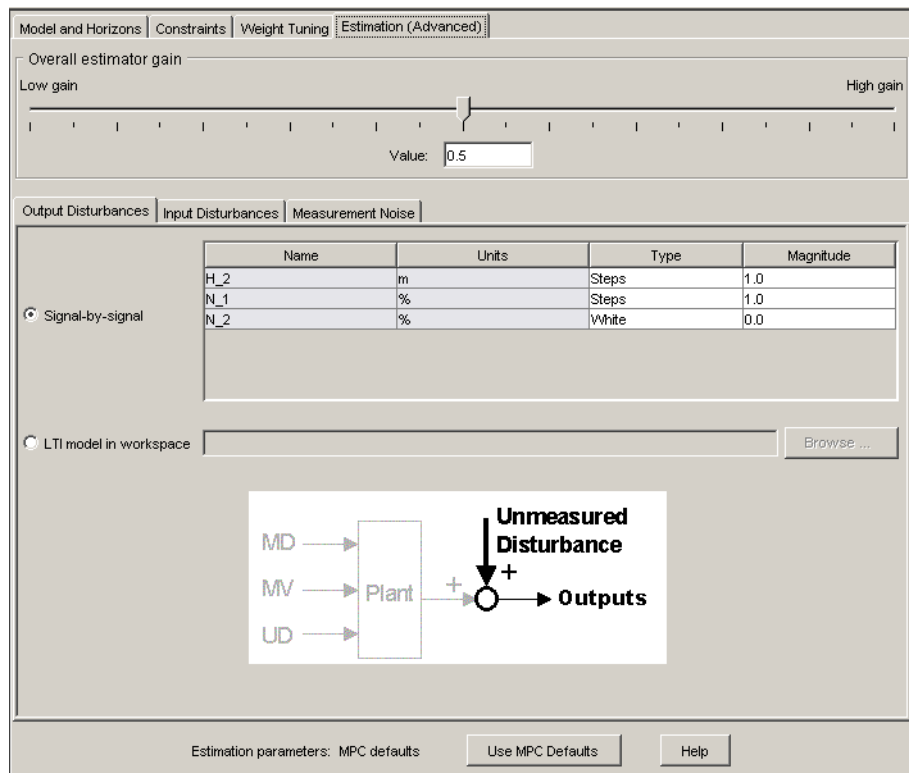
Figure 4-24: Default Estimator Assumptions: Output Disturbances.

Now consider the upper part of the figure. The **Output Disturbances** tab is active, and its **Signal-by-signal** option is selected. According to the tabular data, the controller is assuming independent, step-like disturbances (i.e., integrated white noise) in the first two outputs.

Select the **Input Disturbances** tab. Verify that the controller is also assuming independent step-like disturbances in the unmeasured disturbance input.

Thus, there are a total of three independent, sustained (step-like) disturbances. This allows the controller to eliminate offset in all three measured outputs.

The disturbance magnitudes are unity by default. Making one larger than the rest would signify a more important disturbance at that location.

Select the **Measurement Noise** tab. Verify that white noise (unit magnitude) is being added to each output. The noise magnitude governs how much influence each measurement has on the controller's decisions. For example, if a particular measurement is relatively noisy, the controller will give it less weight, relying instead upon the model predictions of that output. This provides a noise filtering capability.

In the paper machine application, the default disturbance assumptions are reasonable. It is difficult to improve disturbance rejection significantly by modifying them.

Controlling the Nonlinear Plant in Simulink

It's good practice to run initial tests using the linear plant model as described in "Servo Response" on page 4-31 and "Unmeasured Input Disturbance" on page 4-33. Such tests don't introduce prediction error, and are a useful benchmark for more demanding tests with a nonlinear plant model. The controller's prediction model is linear, so such tests introduce prediction error.
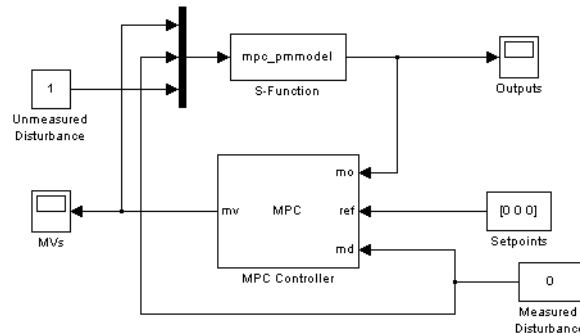
Figure 4-25: Paper Machine Headbox Control Using MPC Tools in Simulink

Figure 4-25 is a Simulink diagram in which the Model Predictive Control Toolbox controller is being used to regulate the nonlinear paper machine headbox model. The block labeled S-Function embodies the nonlinear model, which is coded in an M-file called mpc_pmmodel.m.

As shown in the dialog below, the MPC block references a controller design called MPC1, which was exported to the MATLAB workspace from the design tool. Note also that the measured disturbance inport is enabled, allowing the measured disturbance to be connected as shown in Figure 4-25.
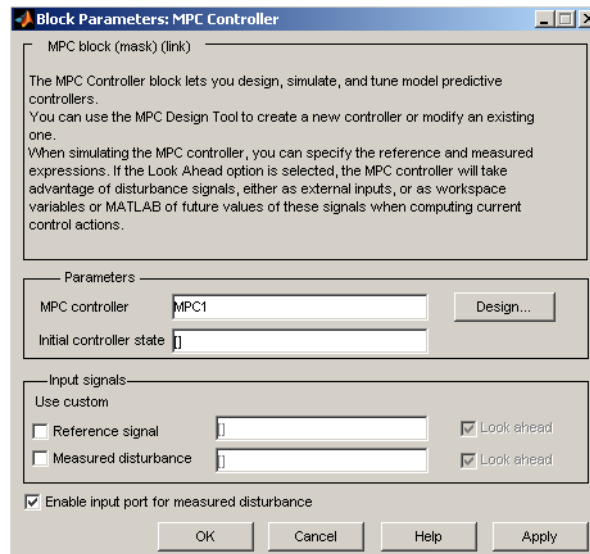

Figure 4-26 shows the scope display from the "Outputs" block for the setup of Figure 4-25, i.e., an unmeasured disturbance. The yellow curve is H_2, the magenta is N_1, and the cyan is N_2. Comparing to Figure 4-23, the results are almost identical, indicating that the effects of nonlinearity and prediction error were insignificant in this case. Figure 4-27 shows the corresponding manipulated variable moves (from the "MVs" scope in Figure 4-25) which are smooth yet reasonably fast.

As disturbance size increases, nonlinear effects begin to appear. For a disturbance size of 4, the results are still essentially the same as shown in Figure 4-26 and Figure 4-27 (scaled by a factor of 4), but for a disturbance size of 6, the setpoint deviations are relatively larger, and the curve shapes differ (not shown). There are marked qualitative and quantitative differences when the disturbance size is 8. When it is 9, deviations become very large, and the MVs saturate. If such disturbances were likely, the controller would have to be retuned to accommodate them.
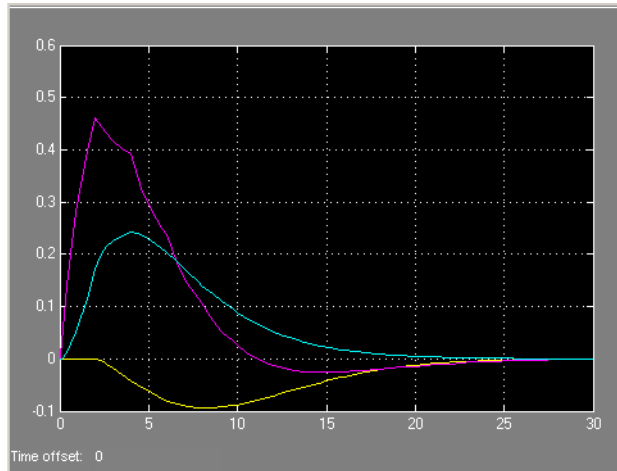
**Figure 4-26: Simulink Test, Output Variables**
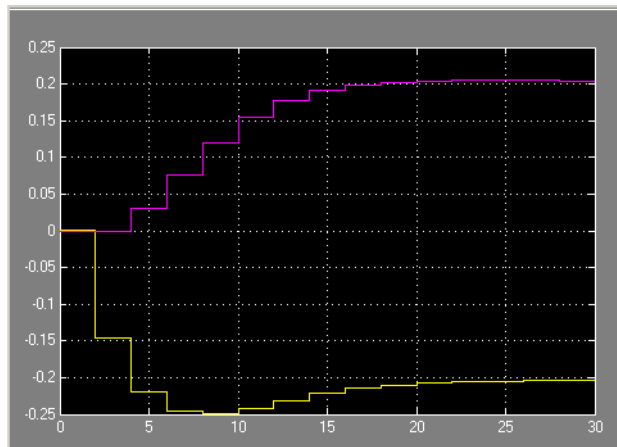
Figure 4-27: Simulink Test, Manipulated Variables

Bumpless Transfer in MPC

During startup of a continuous plant, the operators often set the actuators manually until the plant is near the desired operating point. A sudden switch to automatic control can cause a *bump*, i.e., a large actuator movement. The same issue arises when you try to switch between alternative controllers, each of which has been designed for a particular operating point. A Model Predictive Controller must monitor all known plant signals even when it is not in control of the actuators. This lets it continuously refine its state estimate and make appropriate actuator adjustments when it is switched to automatic. A smooth transition is called bumpless transfer.

A Model Predictive Toolbox demo illustrates this behavior. To initialize the demo, type `mpcbumpless` at MATLAB command line. The Simulink block diagram shown in Figure 4-28 opens.
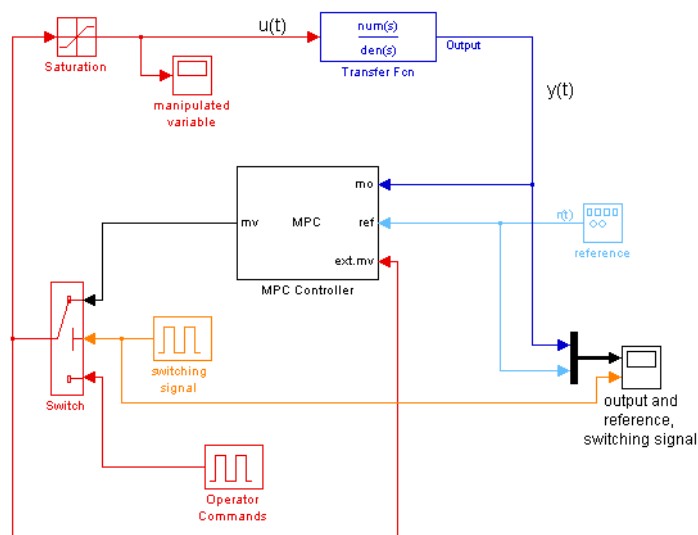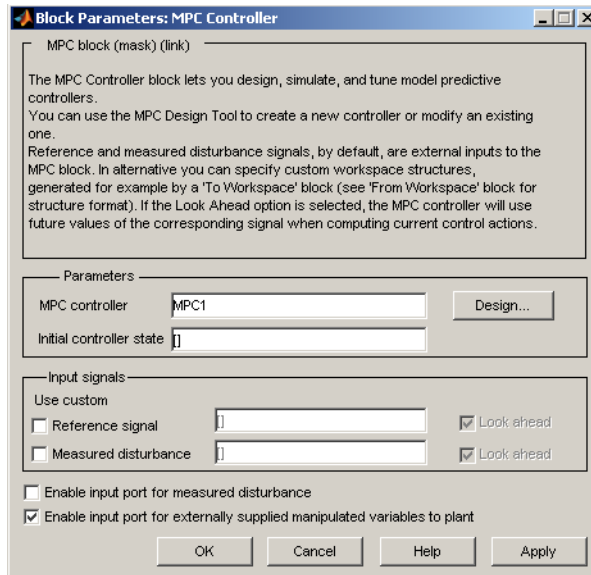

Figure 4-28: Simulink Block Diagram for the MPC Bumpless Transfer Demo

The plant is a stable single-input single-output system. Figure 4-29 shows its open-loop unit step response. Figure 4-30 shows the MPC block configuration settings for this case.
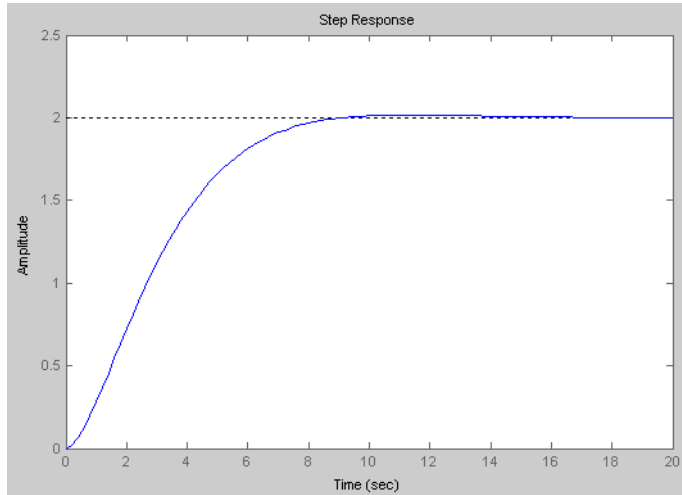
**Figure 4-29: Open-Loop Unit Step Response**

Figure 4-30: MPC Block Configuration Settings

The demo creates MPC1, the controller object. Its sampling period is 0.5. For additional design details, see the demo's documentation.

As shown in Figure 4-30, the block's optional input port for externally supplied manipulated variables is selected. This causes the inport labeled ext.mv to appear (Figure 4-28 shows how this is connnected).

This demo tests the effect of switching the controller from automatic to manual and back. To simulate this, a Pulse Generator block labeled switching signal sends either 1 or 0 to a switch. When it sends 1, the system is in automatic mode, and the MPC block's output goes to the plant. Otherwise, the system is in manual mode, and the signal from the Operator Commands block goes to the plant. In both cases the actual plant input feeds back to the controller, as shown in Figure 4-28, unless the saturation limits of -1 and 1 are encountered. The controller also monitors the plant ouput at all times. As shown in Figure 4-31, the system is in automatic mode for the first 90 time units (switching signal is 1). The controller moves the output from its initial value, 0, to the reference value, -0.5.
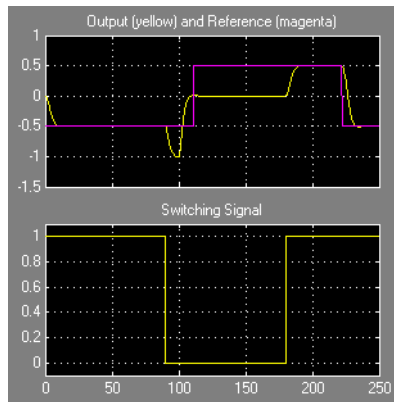
Figure 4-31: Output, Reference and Switching Signal

The following figure shows the required manipulated variable adjustments.
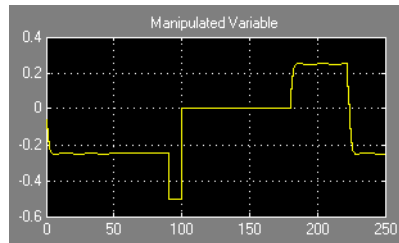
Figure 4-32: Manipulated Variable (Actuator) Adjustments

At time 90, the system switches to manual mode. As shown in Figure 4-32, the operator sets the manipulated variable to -0.5 for 10 time units, and then to 0. Figure 4-31 shows the open-loop response during this period. The controller is in manual mode, so it can't track the reference.

At time 180, the system switches back to automatic mode, where it stays for the duration. Figure 4-31 shows that the output returns to the reference value smoothly, and Figure 4-32 shows similarly smooth adjustments to the manipulated variable.

Note that the controller's state estimator has default zero initial conditions, which are appropriate when this simulation begins. Thus, there is no bump at startup. In general you would need to start the system in manual mode for long enough to allow the controller to acquire an accurate state estimate before switching to automatic mode. You could monitor the controller's adjustments while it was in manual mode to see when it had converged to a steady condition.

Now consider the situation shown in Figure 4-33. The external manipulated variable feedback has been disconnected. The behavior is as if the option were not selected in Figure 4-30, i.e., the block assumes that its adjustments are always going to the plant. This is incorrect whenever the system switches to manual mode.
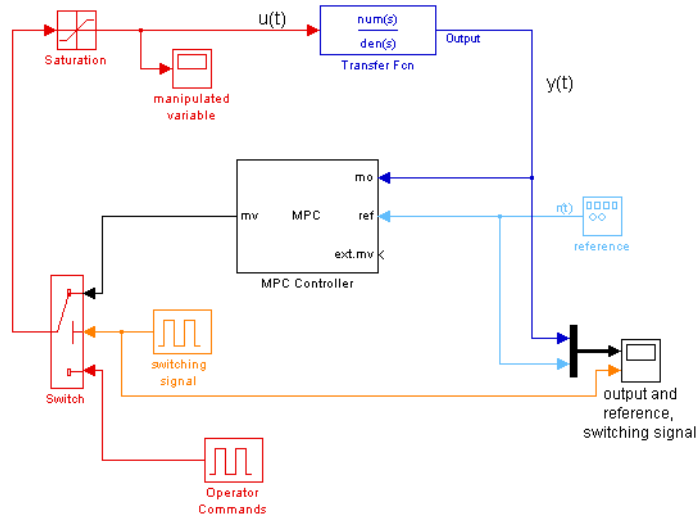
Figure 4-33: External Manipulated Variable Feedback Disconnected

As shown in Figure 4-34 and Figure 4-35, the behavior is identical to the original case for the first 180 time units (compare to Figure 4-31 and Figure 4-32). When the system switches to automatic mode at time 180, however, there is a large bump. This is because the controller is using the incorrect plant input to estimate the plant state during the time the system is in manual mode.
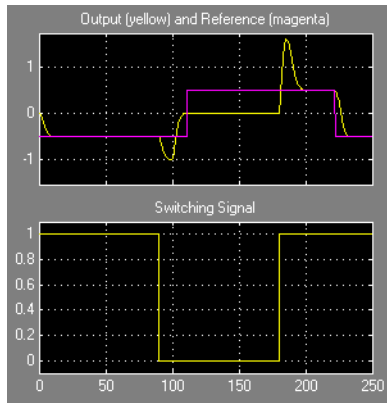
Figure 4-34: Output Response with Manipulated Variable Feedback Disconnected
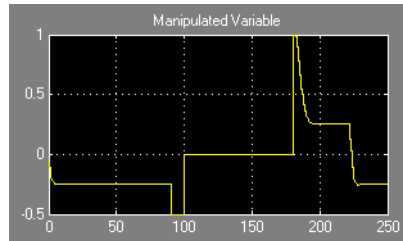
Figure 4-35: Manipulated Variable Adjustments with Manipulated Variable Feedback Disconnected

Reference

[1] Ying, Y., M. Rao, and Y. Sun "Bilinear control strategy for paper making process," *Chemical Engineering Communications* (1992), Vol. 111, pp. 13-28.

5

Design Tool

This chapter is the reference manual for the Model Predictive Control Toolbox design tool (graphical user interface). For example design tool applications, see the Model Predictive Control Toolbox Getting Started manual, or Chapter 4, "Case-Study Examples" in this document.

Opening the MPC Design Tool

To open the Design Tool in MATLAB, type

```
mpctool
```

The design tool is part of the Control and Estimation Tools Manager. When invoked as shown above, the design tool opens and creates a new *project* named **MPCdesign**.

If you started the tool previously, the above command makes the tool visible but doesn't create a new project.

Alternatively, if your Simulink model contains a Model Predictive Controller block, you can double-click on the block to obtain its mask (see example below) and click the **Design** button. If the **MPC controller** field is empty, the design tool will create a default controller. Otherwise, it will load the named controller object, which must be in your workspace, so you can view and modify it.
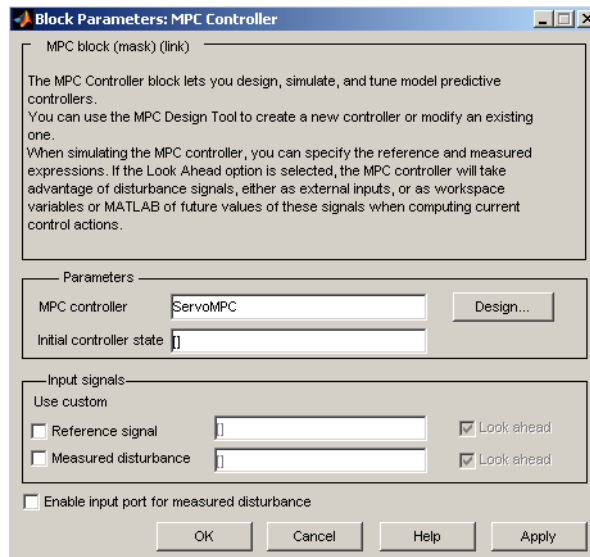
Menu Bar

The design tool's menu bar appears whenever you've selected a Model Predictive Control Toolbox project or task in the tree (see "The Tree View" on page 5-7). The menu bar's MPC option distinguishes it from other control and estimation tools. See the example below. The following sections describe each menu option.
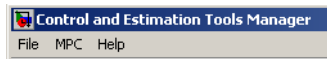
File Menu

New Design

Creates a new (empty) Model Predictive Control Toolbox design project within the Control and Estimation Tools Manager and assigns it a default name. You can also create a new design using the toolbar (see "The Toolbar" on page 5-6).

Load

Loads a saved design. A dialog asks you to specify the MAT-file containing the saved design. If the MAT-file contains multiple projects, you must select the one(s) to be loaded (see example below).
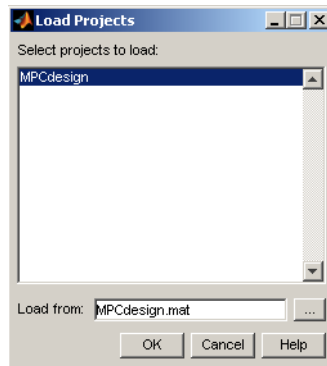
You can also load a design using the toolbar (see "The Toolbar" on page 5-6).

Save

Saves a design so you can use it later. The data are saved in a MAT-file. A dialog allows you to specify the file name (see below). If you are working on multiple projects, you can select those to be saved.
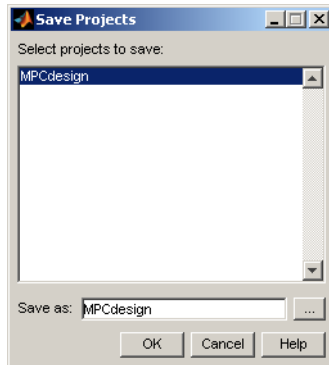
You can also select the **Save** option using the toolbar (see "The Toolbar" on page 5-6.)

Close

Closes the design tool. If you've modified the design, you'll be asked whether or not you want to save it before closing.

MPC Menu

Import

You have the following options:

- **Plant model** – Import a plant model using the model import dialog (see "Importing a Plant Model" on page 5-9)

- **Controller** – Import a controller using the controller import dialog (see "Importing a Controller" on page 5-15)

Export

Export a controller using the export dialog (see "Exporting a Controller" on page 5-19). This option won't be enabled until your design includes at least one fully specified controller.

Simulate

Simulate the *current scenario*, i.e., the one most recently simulated or selected in the tree (see "The Tree View" on page 5-7). You can select this option from the keyboard by typing `Ctrl-R`, or using the toolbar icon (see "The Toolbar" on page 5-6).

The **Simulate** option won't be unabled until your design includes at least one fully specified simulation scenario.

The Toolbar

The toolbar, shown below, provides quick access to certain menu options.
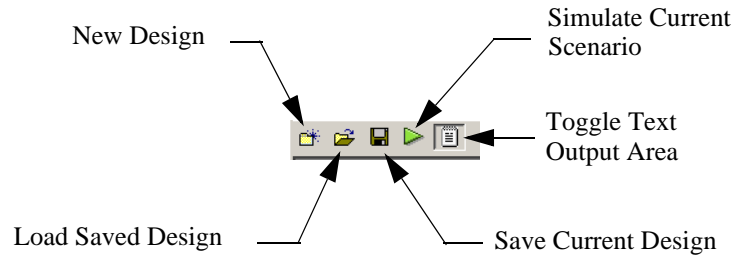
For more information on the first four functions, see the following:

- "New Design" on page 5-3
- "Load" on page 5-3
- "Save" on page 5-4
- "Simulate" on page 5-5

The *text output area* is a text display located along the bottom of the tool that displays progress messages and diagnostics. In the above view, the *toggle* button is pushed in, so the text display area appears. If you are working on a small screen, you might use the toggle button to hide the text area, allowing more room to display the design parameters.

The Tree View

The tree view appears in a frame on the design tool's left-hand side (see example below). When you select one of the tree's *nodes* (by clicking its name or icon) the larger frame to its right shows a dialog pane that allows you to view and edit the specifications associated with that item.

Node Types
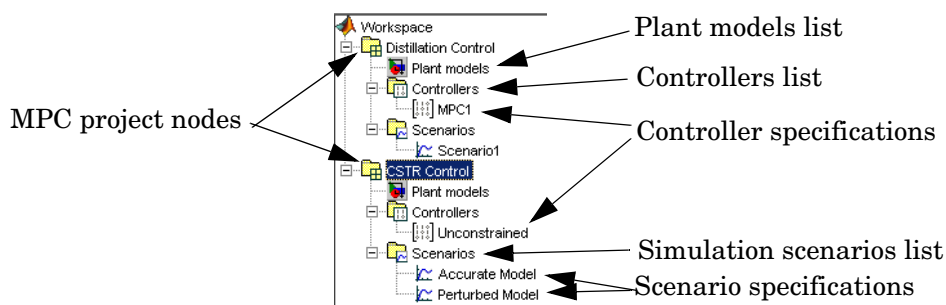
The above example shows two Model Predictive Control Toolbox design project nodes, **Distillation Control** and **CSTR Control**, and their sub-nodes. For more details on each node type, see the following:

- *MPC design project/task* – see "Signal Definition View" on page 5-21
- *Plant models list* – see "Plant Models View" on page 5-26
- *Controllers list* – see "Controllers View" on page 5-29
- *Controller specifications* – see "Simulation Scenarios List" on page 5-33
- Scenarios list – see "Simulation Scenario View" on page 5-59
- Scenario specifications – see "Controller Specifications View" on page 5-36

Renaming a Node

You can rename following node types:

- MPC design project/task

- Controller specifications
- Scenario specifications

To rename a node, do *one* of the following:

- Click on the name, wait for an edit box to appear, type the desired name, and push the Enter key to finalize your choice, OR
- Right-click on the name, select the **Rename** menu option, and enter the desired name in the dialog box, OR
- To rename a controller specification node, select **Controllers** and edit the controller name in the table, OR
- To rename a scenario specification node, select **Scenarios** and edit the scenario name in the table

Importing a Plant Model

To import a plant model, do *one* of the following:

- Select the **MPC/Import/Plant Model** menu option.
- Select the MPC project/task node in the tree (see "The Tree View" on page 5-7), and then click the **Import Plant** button.
- Right-click the MPC project/task node and select the **Import Plant Model** menu option.
- If you've already imported a model, select the **Plant models** node, and then click the **Import** button.
- If you've already imported a model, right-click the **Plant models** node and select the **Import Model** menu option.

All of the above open the **Plant Model Importer** dialog (see dialog box shown in "Import from" on page 5-16 for an example). Within the dialog you can import an LTI model from the workspace or, when you have Simulink Control Design, you can import a linearized plant model from the Simulink model.

The following sections describe the dialog options for importing an LTI model from the workspace.

- "Import from" on page 5-16
- "Import to" on page 5-17
- "Buttons" on page 5-17
- "Importing a Linearized Plant Model" on page 5-12

Import from

Use the radio buttons to set the location from which the model will be imported.

MATLAB Workspace
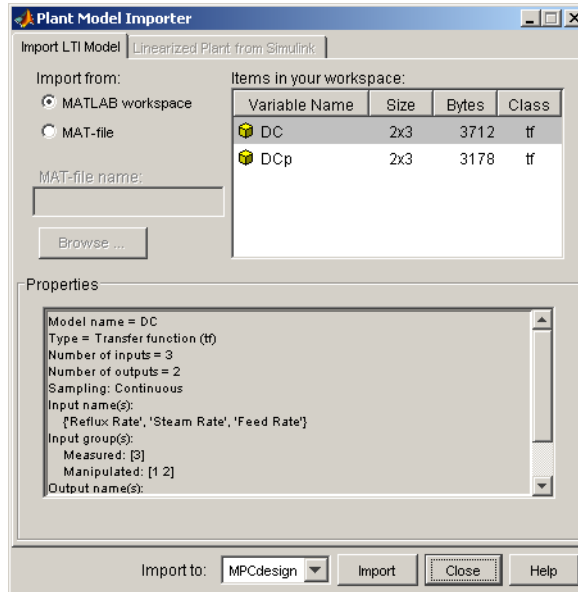
This is the default option and is the case shown in the above example. The **Items in your Workspace** area in dialog's upper-right lists all candidate models in your MATLAB workspace. Select one by clicking on it. The dialog's **Properties** area lists the selected model's properties (the DC model in the above example).

MAT-File

The upper part of the dialog changes as shown below.
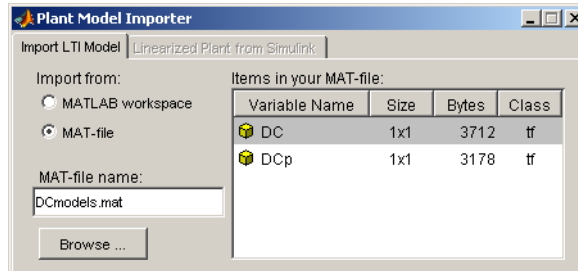
The **MAT-file name** edit box becomes active. Type the desired MAT-file name here (if it's not on the MATLAB path, enter the complete file path). You can also use the **Browse** button which activates a standard file chooser dialog.

In the above example, file DCmodels.mat contains two models. Their names appear in the **Items in your MAT-file** area on the dialog's upper right. As with the workspace option, the selected model's properties appear in the **Properties** area.

Import to

The combo box at the dialog's bottom allows you to specify the MPC project/task into which the plant model will be imported (see example below). It defaults to that most recently active.

Buttons

Import

Select the model you want to import from the **Items** list in the dialog's upper right. Verify that the **Import To** option designates the correct project/task. Click the **Import** button to import the model.

5-11

You can select **Plant models** in the tree to verify that the model has been loaded. (See "The Tree View" on page 5-7, and "Plant Models View" on page 5-26.)

The import dialog remains visible until you close it, allowing you to import additional models.

Close

Click **Close** to close the dialog window. You can also click the close icon on the window's title bar.

Importing a Linearized Plant Model

1 Open the design tool *from within a Simulink model* as discussed in "Opening the MPC Design Tool" on page 5-2.

2 Open the **Plant Model Importer** dialog (see "Importing a Plant Model" on page 5-9).

3 Select the **Linearized Plant from Simulink** pane (see the following example).

Note If you haven't opened the design tool from within a Simulink diagram, you won't be able to access the required pane in step 3.
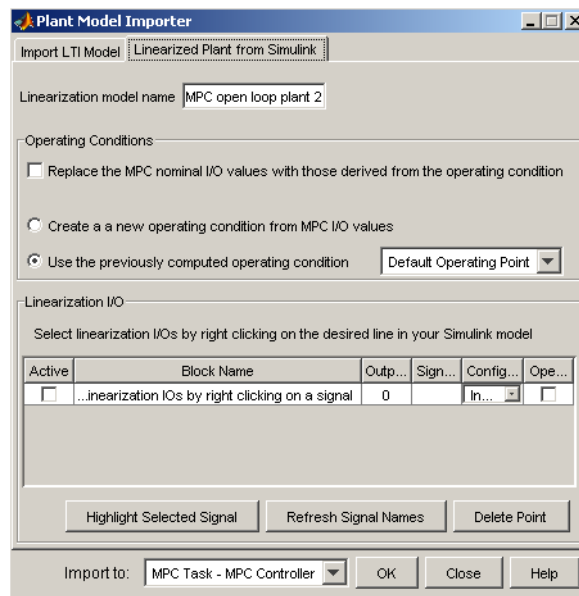
Linearization Process

When you click **OK**, the design tool uses Simulink Control Design to create a linearized plant model. It performs the following tasks automatically:

1 Configures the Control and Estimation Tools Manager.

2 Temporarily inserts linearization input and output points in the Simulink model at the inputs and outputs of the MPC block.

3 When the **Create a new operating condition from MPC I/O values** is selected, the Model Predictive Control Toolbox temporarily inserts output constraints at the inputs/outputs of the MPC block.

4 Finds a steady state operating condition based on the constraints or uses the specified operating condition.

5 Linearizes the plant model about the operating point.

The linearized plant model appears as a new node under **Plant Models**. For details of the linearization process, refer to the Simulink Control Design documentation.

Linearization Options

You can also customize the linearization process in several ways before clicking
OK:

- To specify an alternative name for the linearized plant model, enter the
  name in the **Linearization model name** edit field.
- To use an alternative operating condition, you can
  - Select one from the menu next to **Use the previously computed
    operating condition**. This list contains all operating conditions that exist
    within the current project.
  - Select **Create a new operating condition from MPC I/O values** to
    compute an operating condition by optimization, using the nominal plant
    values as constraints.
- To replace the nominal plant values with the operating point used in the
  linearization, select the check box next to **Replace the MPC nominal I/O
  values with those derived from the operating condition**.
- When there are multiple MPC blocks, use the **Import to** menu to select the
  node within the Control and Estimation Tools Manager that will receive the
  plant model.

In addition, the **Linearization I/O** pane displays the current linearization
input and output points in the model. When creating the linearized model, the
Model Predictive Control Toolbox temporarily modifies these with input and
output points suitable for extracting a linearized plant model.

Importing a Controller

To import a controller, do *one* of the following:

- Select the **MPC/Import/Controller** menu option.
- Select the MPC project/task node in the tree (see "The Tree View" on page 5-7), and then click the **Import Controller** button.
- Right-click the MPC project/task node and select the **Import Controller** menu option.
- If you've already designed a controller, select the **Controllers** node, and then click the **Import** button.
- If you've already designed a controller, right-click the **Controllers** node and select the **Import Controller** menu option.f

All of the above open the MPC Importer dialog. The following sections describe the dialog box options.

- "Import from" on page 5-16
- "Import to" on page 5-17
- "Buttons" on page 5-17

Import from

Use the radio buttons to set the location from which the controller will be imported.

MATLAB Workspace
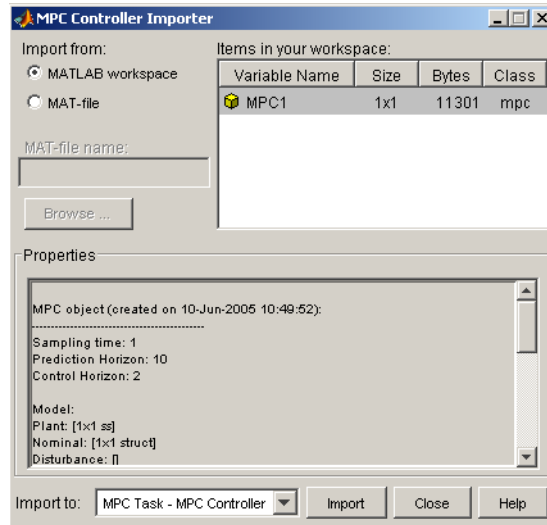
This is the default option and is the case shown in the above example. The **Items in your Workspace** area in dialog's upper-right lists all mpc objects in your workspace. Select one by clicking on it. The **Properties** area lists the properties of the selected model.

MAT-File

The upper part of the dialog changes as shown below.

The **MAT-file name** edit box becomes active. Type the desired MAT-file name here (if it's not on the MATLAB path, enter the complete file path). You can also use the **Browse** button which activates a standard file chooser dialog.

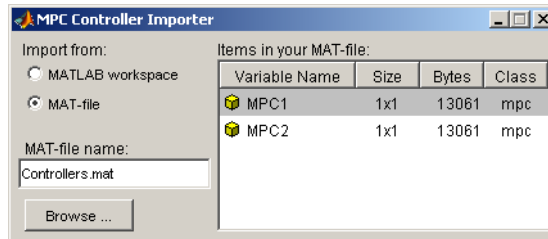In the above example, file Controllers.mat contains two mpc objects. Their names appear in the **Items in your MAT-file** area on the dialog's upper right.

Import to

This allows you to specify the MPC task into which the controller will be imported (see example below). It defaults to that most recently active.

Buttons

Import

Select the controller you want to import from the **Items** list in the dialog's upper right. Verify that the **Import To** option designates the correct project/task. Click the **Import** button to import the controller.

The new controller should appear in the tree as a sub-node of **Controllers**. (See "The Tree View" on page 5-7.)

The imported controller contains a plant model, which appears in the **Plant models** list. (See "Plant Models View" on page 5-26.)

Note If the selected controller is incompatible with any others in the designated project, the design tool will not import it.

Close

Click **Close** to close the dialog window. You can also click the close icon on the window's title bar.

Exporting a Controller

To export a controller, do *one* of the following:

- Select the **MPC/Export** menu option.
- Select **Controllers** in the tree and click its **Export** button.
- In the tree, right-click **Controllers** and select the **Export Controller** menu option.
- In the tree, right-click the controller you wish to export and select the **Export Controller** menu option.

All of the above open the MPC Exporter dialog (see example below). The following sections describe the dialog box options.

- "Dialog Options" on page 5-19
- "Buttons" on page 5-20

Dialog Options

The following sections describe the dialog's options.

Controller Source

Use this to select the project/task containing the controller to be exported. It defaults to the project/task most recently active.

Controller to Export

Use this to specify the controller to be exported. It defaults to the controller most recently selected in the tree.

Name to Assign

Use this to assign a valid MATLAB variable name (no spaces). It defaults to the selected controller's name (with spaces removed, if any).

Export to Workspace

Select this radio button if you want the controller to be exported to the MATLAB workspace.

Export to MAT-file

Select this radio button if you want the controller to be exported to a MAT-file.

Buttons

Export

If you've selected the **to Workspace** option, clicking **Export** causes a new mpc object to be created in your MATLAB workspace. (If one having the assigned name already exists, you'll be asked if you want to overwrite it.) You can use the MATLAB whos command to verify that the controller has been exported.

If you've selected the **to MAT-file** option, clicking **Export** opens a standard file chooser that allows you to specify the file.

In either case, the dialog window remains visible, allowing you to export additional controllers.

Close

Click **Close** to close the dialog window. You can also click the close icon on the window's title bar.

Signal Definition View

The signal definition view appears whenever you select a Model Predictive Control Toolbox project/task node in the tree (see "The Tree View" on page 5-7). It is also the view you'll see when you open the design tool for the first time. An example appears below.
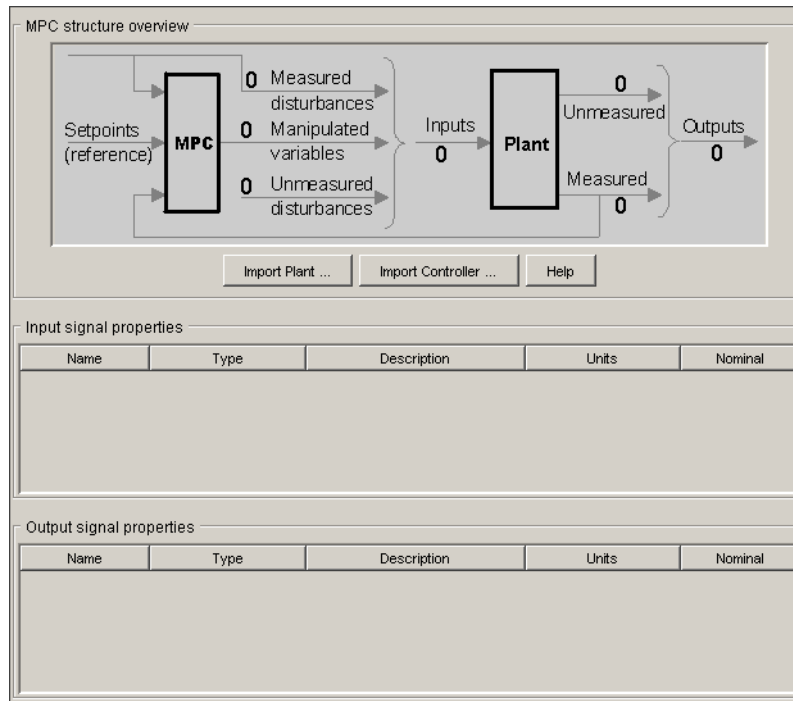

The following sections describe the view's main features:

- "MPC Structure Overview" on page 5-22
- "Buttons" on page 5-22
- "Signal Properties Tables" on page 5-22
- "Right-Click Menu Options" on page 5-24

MPC Structure Overview

This upper section is a non-editable display of your application's structure. Once you've imported a plant model (or controller), the graphic shows counts for the five possible signal types, as in the example below.
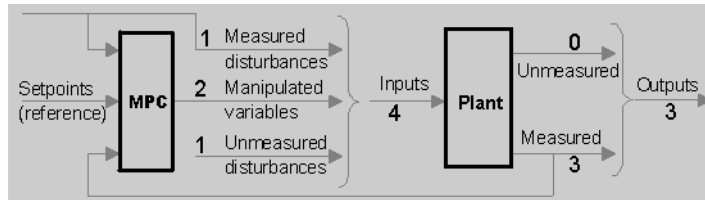
The counts will change if you edit the signal types.

Buttons

Import Plant

Clicking this opens the Plant Model Importer dialog (see "Importing a Plant Model" on page 5-9).

Import Controller

Clicking this opens the MPC Importer dialog (see "Importing a Controller" on page 5-15).

Note You won't be allowed to proceed with your design until you import a plant model. You can do so indirectly by importing a controller or loading a saved project.

Signal Properties Tables

Two tables display the properties of each signal in your design.

Input Signal Properties

The plant's input signals appear as table rows (see example below).

Name	Type	Description	Units	Nominal
G_p	Manipulated	Feed flow rate	kg/h	0.0
G_w	Manipulated	White water flow rate	kg/h	0.0
N_p	Meas. disturb.	Feed consistency	%	0.0
N_w	Unmeas. disturb.	White water consistency	%	0.0

Input signal properties

The table's columns are editable and have the following significance:

- **Name** – The signal name, an alphanumeric string used to label other tables, graphics, etc. Each name must be unique. The design tool assigns a default name if your imported plant model doesn't specify one.

- **Type** – One of the three valid Model Predictive Control Toolbox input signal types. The above example shows one of each. To change a signal's type, click on the table cell and select the desired type from the resulting menu. The valid signal types are as follows:

   Manipulated – A signal that will be manipulated by the controller, i.e., an actuator (valve, motor, etc.)

   Measured Disturbance – An independent input whose value is measured and used as a controller input for *feedforward compensation*

   Unmeasured Disturbance – An independent input representing an unknown, unexpected disturbance.

- **Description** – An optional descriptive string.

- **Units** – Optional units (dimensions), a string. Used to label other dialogs, simulation plots, etc.

- **Nominal** – The signal's nominal value. The design tool defaults this to zero. Any value you assign here will be the default initial condition in simulations.

Note Your design must include at least one *manipulated variable*. The other input signal types need not be included.

Output Signal Properties
The plant's output signals appear as table rows (see example below).

Output signal properties

Name	Type	Description	Units	Nominal
H_2	Measured	Headbox level	m	0.0
N_1	Measured	Feed tank consistency	%	0.0
N_2	Measured	Head box consistency	%	0.0

The table's columns are editable and have the following significance:

- **Name** – The signal name, an alphanumeric string used to label other tables, graphics, etc. Each name must be unique. The design tool assigns a default name if your imported plant model doesn't specify one.

- **Type** – One of the two valid Model Predictive Control Toolbox output signal types. The above example shows one of each. To change a signal's type, click on the table cell and select the desired type from the resulting menu. The valid signal types are as follows:

  Measured – A signal the controller can use for feedback

  Unmeasured – Predicted by the plant model but unmeasured. Can be used as an indicator. Can also be assigned a setpoint or constrained.

- **Description** – An optional descriptive string.

- **Units** – Optional units (dimensions), a string. Used to label other dialogs, simulation plots, etc.

- **Nominal** – The signal's nominal value. The design tool defaults this to zero. Any value you assign here will be the default initial condition in simulations.

Note Your design must include at least one *measured* output. Inclusion of unmeasured outputs is optional.
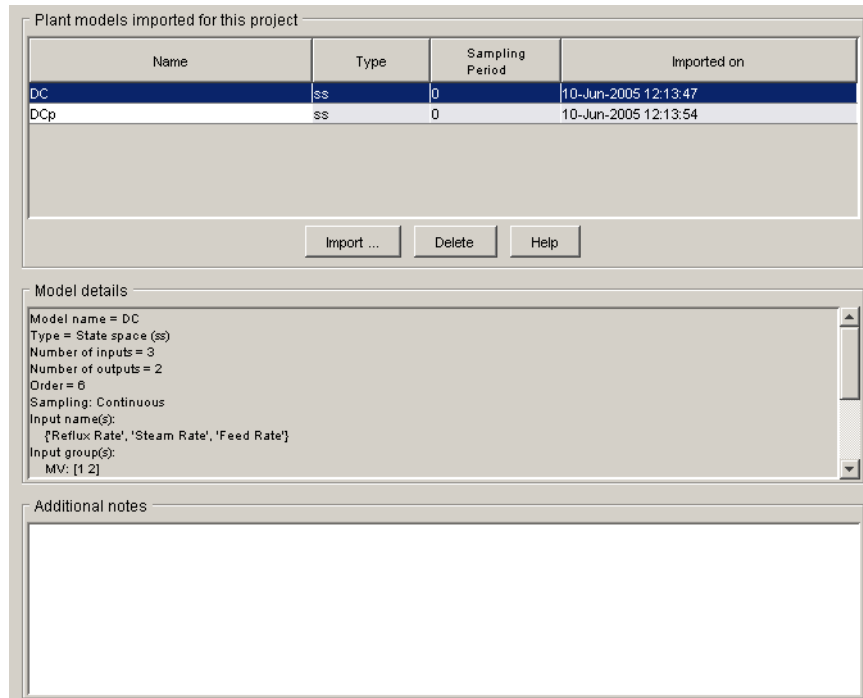

Right-Click Menu Options

Right-clicking on an MPC project/task node allows you to choose one of the following menu items:

- **Import Plant Model** – Opens the Plant Model Importer dialog (see "Importing a Plant Model" on page 5-9)

- **Import Controller** – Opens the MPC Importer dialog (see "Importing a Controller" on page 5-15)

- **Clear Project** – Erases all plant models, controllers, and scenarios in your design, returning the project to its initial empty state.

- **Delete Project** – Deletes the selected project node.

Plant Models View

Selecting **Plant models** in the tree dispays this view (see example below).

The following sections describe the view's main features.

- "Plant Models List" on page 5-27
- "Model Details" on page 5-27
- "Additional Notes" on page 5-28
- "Buttons" on page 5-28
- "Right-Click Options" on page 5-28

Plant Models List

This table lists all the plant models you've imported and/or plant models contained in controllers that you've imported. The example below lists two imported models, DC and DCp.

The **Name** field is editable. Each model must have a unique name. The name you assign here will be used within the design tool, but will not alter the original model's name.

The **Type** field is noneditable and indicates the model's LTI object type (see the Control System Toolbox documentation for a detailed discussion of LTI models).

The **Sampling Period** field is zero for continuous-time models, and a positive real value for discrete-time models.

The **Imported on** field gives the date and time the model was imported into the design tool.

Model Details

This scrollable viewport shows details of the model currently selected in the plant models list (see "Plant Models List" on page 5-27). An example appears below.
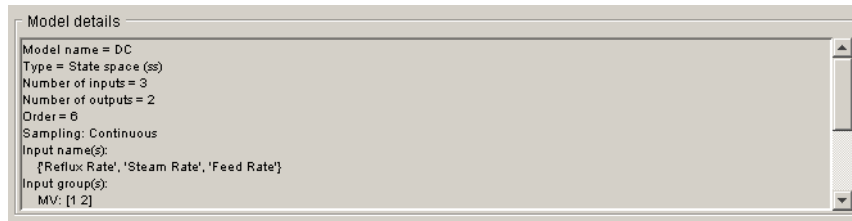
Additional Notes

You can use this editable text area to enter comments, distinguishing model features, etc.

Buttons

Import

Opens the Plant Model Importer dialog (see "Importing a Plant Model" on page 5-9).

Delete

Deletes the selected model. If the model is being used elsewhere (i.e., in a controller or scenario), the first model in the list replaces it (and a warning message appears).
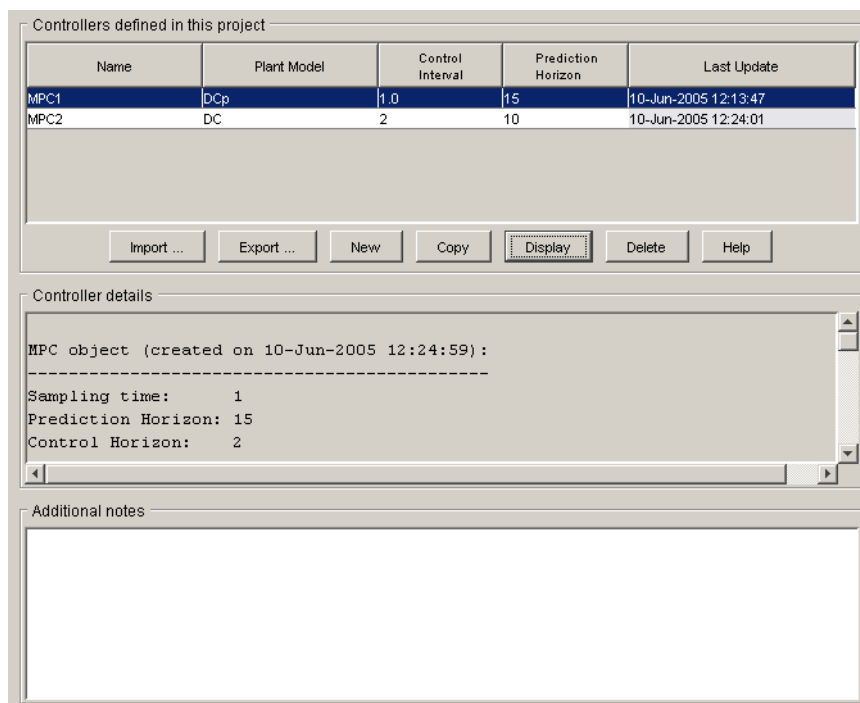
Right-Click Options

Right-clicking on the **Plant models** node causes the following menu option to appear.

Import Model

Opens the Plant Model Importer dialog (see "Importing a Plant Model" on page 5-9).

Controllers View

Selecting **Controllers** in the tree displays this view (see example below).

The following sections describe the view's main features.

Controllers List

This table lists all the controllers in your project. The example below lists two controllers, MPC1 and MPC2.

The **Name** field is editable. The name you assign here must be unique. You will refer to it elsewhere in the design tool, e.g., when you use the controller in a simulation scenario. Each listed controller corresponds to a subnode of **Controllers** (see "The Tree View" on page 5-7). Editing the name in the table will rename the corresponding subnode.

The **Plant Model** field is editable. To change the selection, click on the cell and choose one of your models from the list. (All models appearing in the Plant Models View are valid choices. See "Plant Models View" on page 5-26.)

The **Control Interval** field is editable and must be a positive real number. You can also set it in the Controller Specifications view (see "Model and Horizons Tab" on page 5-37 for more details).

The **Prediction Horizon** field is editable and must be a positive, finite integer. You can also set in the Controller Specifications view (see "Model and Horizons Tab" on page 5-37 for more details).

The noneditable **Last Update** field gives the date and time the controller was most recently modified.

Controller Details

This scrollable viewport shows details of the controller currently selected in the controllers list (see "Controllers List" on page 5-30). An example appears below.
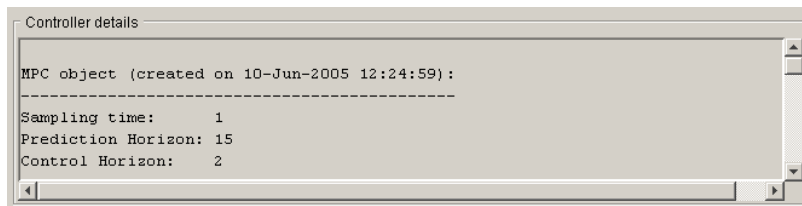
Note This view shows controller details once you have used the controller in a simulation. Prior to that, it is empty. If necessary, you can use the **Display** button to force the details to appear.

Additional Notes

You can use this editable text area to enter comments, distinguishing controller features, etc.

Buttons

Import

Opens the MPC Importer dialog (see "Importing a Controller" on page 5-15).

Export

Opens the MPC Exporter dialog (see "Exporting a Controller" on page 5-19).

New

Creates a new controller specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Copy

Copies the selected controller, creating a controller specification subnode containing the same controller settings, and assigning it a default name.

Display

Calculates and displays details for the selected controller.

Delete

Deletes the selected controller. If the controller is being used elsewhere (i.e., in a simulation scenario), the first controller in the list replaces it (and a warning message appears).

Right-Click Options

Right-clicking on the **Controllers** node causes the following menu options to appear.

New Controller

Creates a new controller specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Import Controller

Opens the MPC Importer dialog (see "Importing a Controller" on page 5-15).

Export Controller

Opens the MPC Exporter dialog (see "Exporting a Controller" on page 5-19).

Simulation Scenarios List

Selecting **Scenarios** in the tree causes this view to appear (see example below).
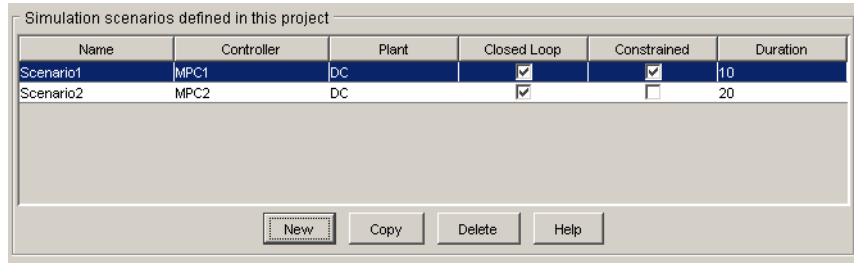

The following sections describe the view's main features.

- "Scenarios List" on page 5-34
- "Scenario Details" on page 5-35
- "Additional Notes" on page 5-35
- "Buttons" on page 5-35
- "Right-Click Options" on page 5-35

Scenarios List

This table lists all the scenarios in your project. The example below lists two, Scenario1 and Scenario2.

Name	Controller	Plant	Closed Loop	Constrained	Duration
Scenario1	MPC1	DC	☑	☑	10
Scenario2	MPC2	DC	☑	☐	20

Simulation scenarios defined in this project

[New] [Copy] [Delete] [Help]

The **Name** field is editable. The assigned name must be unique. Each listed scenario corresponds to a subnode of **Scenarios** (see "The Tree View" on page 5-7). Editing the name in the table will rename the corresponding subnode.

The **Controller** field is editable. To change the selection, click on the cell and choose one of your controllers from the list. (All controllers appearing in the Controllers View are valid choices. See "Controllers View" on page 5-29). You can also set this using the Scenario Specifications view (for more discussion, see "Simulation Scenario View" on page 5-59).

The **Plant** field is editable. To change the selection, click on the cell and choose one of your plant models from the list. (All models appearing in the Plant Models View are valid choices. See "Plant Models View" on page 5-26). You can also set this in the scenario specifications (for more discussion, see "Simulation Scenario View" on page 5-59).

The **Closed Loop** field is an editable checkbox. If unchecked, the simulation will be open loop. You can also set it in the scenario specifications (for more discussion see "Simulation Scenario View" on page 5-59).

The **Constrained** field is an editable checkbox. If unchecked, the simulation will ignore all constraints specified in the controller design. You can also set it in the scenario specifications (for more discussion see "Simulation Scenario View" on page 5-59).

The **Duration** field is editable and must be a positive, finite real number. It sets the simulation duration. You can also set it in the scenario specifications (for more discussion see "Simulation Scenario View" on page 5-59).

Scenario Details

This area is blank at all times.

Additional Notes

You can use this editable text area to enter comments, distinguishing scenario features, etc.

Buttons

New

Creates a new scenario specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Copy

Copies the selected scenario, creating a scenario specification subnode containing the same settings, and assigning it a default name.

Delete

Deletes the selected scenario.

Right-Click Options

Right-clicking on the **Scenarios** node causes the following menu option to appear

New Scenario

Creates a new scenario specification subnode containing the default Model Predictive Control Toolbox settings, and assigns it a default name.

Controller Specifications View

This view appears whenever you select one of your controller specification nodes (see "The Tree View" on page 5-7). It allows you to specify or review controller settings. It consists of four tabs, each devoted to a particular design aspect. All settings have default values, but these might not be best for your application.

The following sections describe the view's main features.

- "Model and Horizons Tab" on page 5-37
- "Constraints Tab" on page 5-40
- "Constraint Softening" on page 5-42
- "Weight Tuning Tab" on page 5-47
- "Estimation Tab" on page 5-50
- "Right-Click Menus" on page 5-57

Model and Horizons Tab
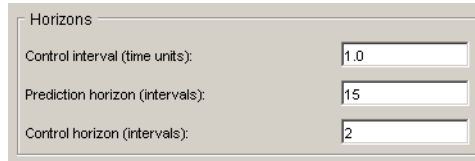

Plant Model

This combo box allows you to specify the plant model the controller uses for its predictions. You can choose any of the plant models you've imported. (See "Importing a Plant Model" on page 5-9.)
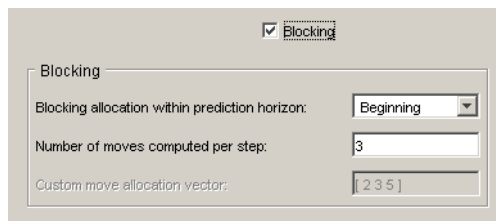
Horizons

The **Control interval** sets the elapsed time between successive controller moves. It must be a positive, finite real number. The calculations assume a zero-order hold on the manipulated variables (the signals adjusted by the controller). Thus, these signals are constant between moves.

The **Prediction horizon** sets the number of *control intervals* over which the controller predicts its outputs when computing controller moves. It must be a positive, finite integer.

The **Control horizon** sets the number of moves computed. It must be a positive, finite integer, and must not exceed the prediction horizon. If less than the prediction horizon, the final computed move fills the remainder of the prediction horizon.

For more discussion, see "A Typical Sampling Instant" on page 1-5, and "Prediction and Control Horizons" on page 1-8.

Blocking

By default, the **Blocking** option is unchecked (off). When on as shown above, the design tool replaces the **Control horizon** specification (see "Horizons" on page 5-38) with a move pattern determined by the following settings:
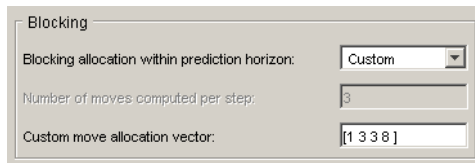
- **Blocking allocation within prediction horizon** – choices are

  Beginning – successive moves at the beginning of the prediction horizon, each with a duration of one control interval

  Uniform – the prediction horizon is divided by the number of moves and rounded to obtain an integer duration, and each computed move has this duration (the last move extends to fill the prediction horizon)

  End – successive moves at the end of the prediction horizon, each with a duration of one control interval

  Custom – you specify the duration of each computed move

- **Number of moves computed per step** – the number of moves computed when the allocation setting is **Beginning**, **Uniform**, or **End**. Must be a positive integer not exceeding the prediction horizon.

- **Custom move allocation vector** – the duration of each computed move, specified as a row vector. In the example below, there are 4 moves, the first lasting 1 control interval, the next two lasting 3, and the final lasting 8 for a total of 15. The **Number of moves computed per step** setting is disabled (ignored).

The sum of the vector elements should equal the prediction horizon (15 in this case). If not, the last move is extended or truncated automatically.

Note When **Blocking** is off, the controller uses the **Beginning** allocation with **Number of moves computed per step** equal to the **Control horizon**.

For more discussion, see "Blocking" on page 1-14.

Constraints Tab

This pane allows you to specify *constraints* (bounds) on *manipulated variables* and *outputs*. Constraints can be *hard* or *soft*. By default, all variables are *unconstrained*, as shown in the view below.

Note If you specify constraints, manipulated variable constraints are hard by default, whereas output variable constraints are soft by default. You can customize this behavior, as discussed in the following sections. For additional information on constraints, see "Optimization and Constraints" on page 1-10, and "Optimization Problem" on page 2-5.

Constraints on Manipulated Variables

The example below is for an application with two manipulated variables (MVs), each represented by a table row.

Name	Units	Minimum	Maximum	Max Down Rate	Max Up Rate
Reflux Rate		0	85	-10	10
Steam Rate		0	52		

Constraints on manipulated variables

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The remaining table columns are editable. If you leave a cell blank, the controller ignores that constraint. You can achieve the same effect by entering -Inf (for a **Minimum** or **Max down rate**) or Inf (for a **Maximum** or **Max up rate**).

The **Minimum** and **Maximum** values set each MV's range.

The **Max down rate** and **Max up rate values** set the amount the MV can change *in a single control interval*. The **Max down rate** must be negative or zero. The **Max up rate** must be positive or zero.

Constraint values must be consistent with your nominal values (see "Input Signal Properties" on page 5-22). In other words, each MV's nominal value must satisfy the constraints.

Constraint values must also be self-consistent. For example, an MV's lower bound must not exceed its upper bound.

Constraints on Output Variables

The example below is for an application with two output variables, each represented by a table row.

Constraints on output variables			
Name	Units	Minimum	Maximum
Distillate Purity			
Bottoms Purity			

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The remaining table columns are editable. If you leave a cell blank (as above), the controller ignores that constraint. You can achieve the same effect by entering -Inf (for a **Minimum**) or Inf (for a **Maximum**).

Constraint values must be consistent with your nominal values (see "Output Signal Properties" on page 5-23). In other words, each output's nominal value must satisfy the constraints.

Constraint values must also be self-consistent. For example, an output's lower bound must not exceed its upper bound.

Note Don't constrain outputs unless this is an essential aspect of your application. It is usually better to define output setpoints (reference values) rather than constraints.

Constraint Softening

A *hard* constraint cannot be violated. Hard constraints are risky, especially for outputs, because the controller will ignore its other objectives in order to satisfy

them. Also, the constraints might be impossible to satisfy in certain situations, in which case the controllers calculations are mathematically *infeasible*.

The Model Predictive Control Toolbox allows you to specify *soft* constraints. These can be violated, but you specify a violation tolerance for each constraint (the *relaxation band*). See the example specifications below.

MPC Constraint Softening

Specify relaxation bands

Input constraints

Name	Units	Minimum	Min Band	Maximum	Max Band	Max Down...	Max Down...	Max Up Rate	Max Up Ba...
Reflux Rate		0		85		-10		10	
Steam Rate		0		52					

Output constraints

Name	Units	Minimum	Min Band	Maximum	Max Band
Distillate Purity		90			
Bottoms Purity					

Overall constraint softness

Soft constraints   Hard constraints

Value: 0.75

OK   Cancel   Help

To open this dialog, click the **Constraint softening** button at the bottom of the **Contraints** tab in the controller specification view (see "Constraints Tab" on page 5-40).

Input Constraints

An example input constraint softening specification appears below.

Name	Units	Minimum	Min Band	Maximum	Max Band	Max Down...	Max Down...	Max Up Rate	Max Up Ba...
Reflux Rate		0		85		-10		10	
Steam Rate		0	2	52	2				

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Minimum**, **Maximum**, **Max down rate**, and **Max up rate** columns are editable. Their values are the same as on the main **Constraints** tab (see "Constraints on Manipulated Variables" on page 5-41). You can specify them in either location.

The remaining columns specify the *relaxation band* for each constraint. An empty cell is equivalent to a zero, i.e., a hard constraint.

Entries must be zero or positive real numbers. To soften a constraint, increase its relaxation band.

The example above shows a relaxation band of 2 moles/min for the steam flow rate's lower and upper bounds. The lack of a relaxation band setting for the reflux flow rate's constraints means that these will be hard.

Note The relaxation band is a relative tolerance, not a strict bound. In other words, the actual constraint violation can exceed the relaxation band.

Output Constraints

An example output constraint specification appears below.

Name	Units	Minimum	Min Band	Maximum	Max Band
Distillate Purity		90	0.5		
Bottoms Purity		93	2		

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Minimum** and **Maximum** columns are editable. Their values are the same as on the main **Constraints** tab (see "Constraints on Output Variables" on page 5-42). You can specify them in either location.

The remaining columns specify the *relaxation band* for each constraint. An empty cell is equivalent to 1.0, i.e., a *soft* constraint.

Entries must be zero or positive real numbers. To soften a constraint, increase its relaxation band.

The example above shows a relaxation band of 0.5 Mole % for the distillate purity lower bound, and a relaxation band of 2 Mole % for the bottoms purity lower bound (the softer of the two constraints).

Note The relaxation band is a relative tolerance, not a strict bound. In other words, the actual constraint violation can exceed the relaxation band.

Overall Constraint Softness

The relaxation band settings allow you to adjust the hardness/softness of each constraint. You can also soften/harden all constraints simultaneously using the slider at the bottom of the dialog pane.
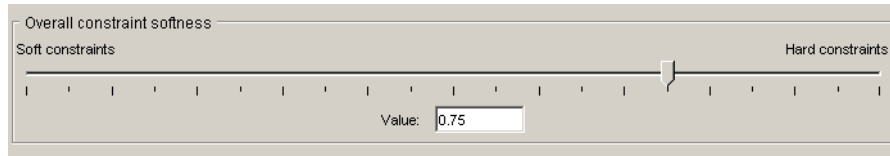
You can move the slider or edit the value in the edit box, which must be between 0 and 1.

Buttons

OK – Closes the constraint softening dialog, implementing changes to the tabular entries or the slider setting.

Cancel – Closes the constraint softening dialog without changing anything.

Weight Tuning Tab

The example below shows the Model Predictive Control Toolbox default tuning weights for an application with two manipulated variables and two outputs.

The following sections discuss the three pane areas in more detail. For additional information, see "Optimization Problem" on page 2-5.

Input Weights

Name	Description	Units	Weight	Rate Weight
Reflux Rate	Molar reflux rate	kmol/min	0	0.1
Steam Rate	Steam heating rate	kmol/min	0	0.1

The **Name**, **Description**, and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Weight** column sets a penalty on deviations of each manipulated variable (MV) from its *nominal value*. The weight must be zero or a positive real number. The default is zero, meaning that the corresponding MV can vary freely provided that it satisfies its constraints (see "Constraints on Manipulated Variables" on page 5-41).

A large **Weight** discourages the corresponding MV from moving away from its nominal value. This can cause *steady state error* (offset) in the output variables unless you have extra MVs at your disposal.

Note To set the nominal values, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Rate Weight** value sets a penalty on MV *changes*, i.e., on the magnitude of each MV move. Increasing the penalty on a particular MV causes the controller to change it more slowly. The table entries must be zero or positive real numbers. These values have no effect in steady state.

Output Weights

Output weights			
Name	Description	Units	Weight
Distillate Purity	Distillate product purity	mol %	1.0
Bottoms Purity	Bottoms product purity	mol %	1.0

The **Name**, **Description**, and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Weight** column sets a penalty on deviations of each output variable from its *setpoint* (or *reference) value*. The weight must be zero or a positive real number.

A large **Weight** discourages the corresponding output from moving away from its setpoint.

If you don't need to hold a particular output at a setpoint, set its **Weight** to zero. This may be the case, for example, when an output doesn't have a target value and is being used as an indicator variable only.

Overall (Slider Control)

Overall	
More robust	Faster response

Value: 0.8
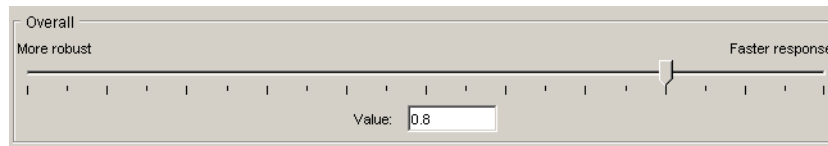
The slider adjusts the weights on all variables simultaneously. Moving the slider to the left increases rate penalties relative to setpoint penalties, which often (but not always!) increases controller robustness. The disadvantage is that disturbance rejection and setpoint tracking become more sluggish.

You can also change the value in the edit box. It must be a real number between 0 and 1. The actual effect is nonlinear. You will generally need to run trials to determine the best setting.

Estimation Tab

Use these specifications to shape the controller's response to unmeasured disturbances and measurement noise.

The example below shows the Model Predictive Control Toolbox default settings for an application with two output variables and no unmeasured disturbance inputs.

The following sections cover each estimation feature in detail. For additional information, see "State Estimation" on page 1-13 for an introduction, and "State Estimation" on page 2-8 for detailed information.

Button (MPC Default Settings)

If you edit any of the Estimation tab settings, the display near the top will appear as follows.

Estimation parameters: user-specified   [Use MPC Defaults]

To return the settings to the default state, click the Use MPC defaults button, causing the display to revert to the default condition shown below.

Estimation parameters: MPC defaults   [Use MPC Defaults]

Overall Estimator Gain

Overall estimator gain
Low gain   High gain

Value: [0.5]
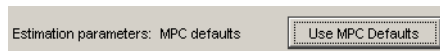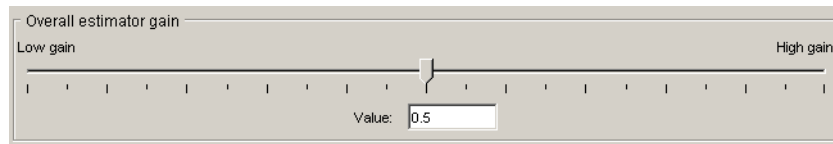
This slider determines the controller's overall disturbance response. As you move the slider to the left, the controller responds less aggressively to unexpected changes in the outputs, i.e., it assumes that such changes are more likely to be caused by measurement noise rather than a *real* disturbance.

You can also change the value in the edit box. It must be between zero and 1. The effect is nonlinear, and you might need to run trial simulations to achieve the desired result.

Output Disturbances

Use these settings to model unmeasured disturbances adding to the plant outputs.

The example below shows the tab's appearance with the **Signal-by-signal** option selected for an application having two plant outputs.
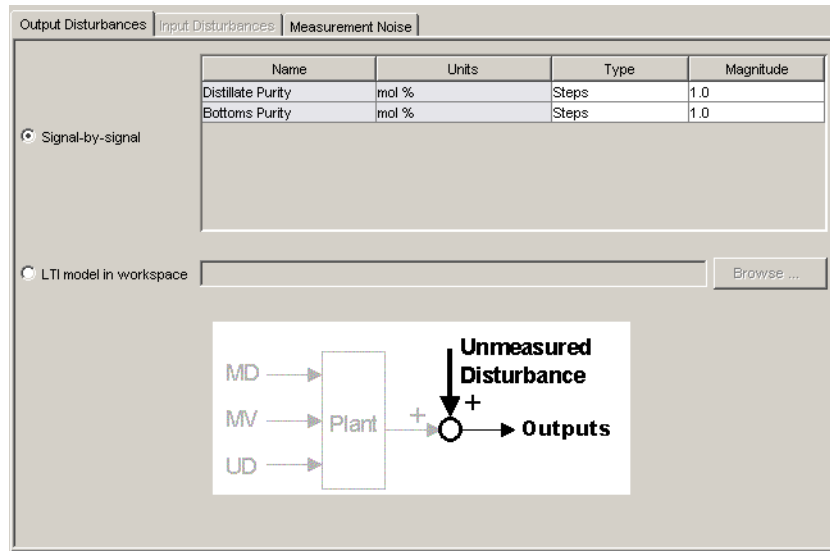
The graphic shows the disturbance location.

Use the table to specify the disturbance character for each output.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Type** column sets the disturbance character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **Steps** – simulates random step-like disturbances (integrated white noise)
- **Ramps** – simulates a random drifting disturbance (doubly-integrated white noise)
- **White** – white noise

The **Magnitude** column specifies the standard deviation of the white noise assumed to create the disturbance. Set it to zero if you want to turn off a particular disturbance.

For example, if **Type** is **Steps** and **Magnitude** is 2, the disturbance model is integrated white noise, where the white noise has a standard deviation of 2.

If these options are too restrictive, select the **LTI model in Workspace** option. The tab appearance changes to the view shown below.
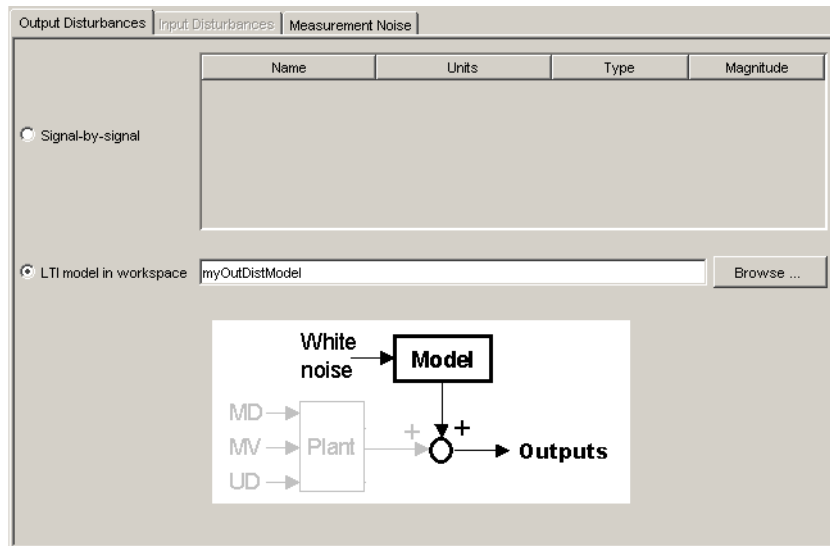
You must specify an LTI output disturbance model residing in your workspace. The **Browse** button opens a dialog listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

The model must have the same number of outputs as the plant.

The white noise entering the model is assumed to have unity standard deviation.

Input Disturbances

Use these settings to model disturbances affecting the plant's unmeasured disturbance inputs.

Note This option is available only if your plant model includes unmeasured disturbance inputs.

The example below shows the tab's appearance with the **Signal-by-signal** option selected for a plant having one unmeasured disturbance input. The graphic shows the disturbance location.

Use the table to specify the character of each unmeasured disturbance input.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Type** column sets the disturbance character. To edit this, click the cell and select from the resulting menu. You have the following options:

- **Steps** – simulates random step-like disturbances (integrated white noise)
- **Ramps** – simulates a random drifting disturbance (doubly-integrated white noise)
- **White** – white noise

The **Magnitude** column specifies the standard deviation of the white noise assumed to create the disturbance. Set it to zero if you want to turn off a particular disturbance.

For example, if **Type** is **Steps** and **Magnitude** is 2, the disturbance model is integrated white noise, where the white noise has a standard deviation of 2.

If the above options are too restrictive, select the **LTI model in Workspace** option. The tab appearance changes to the view shown below.

You must specify an LTI disturbance model residing in your workspace. The **Browse** button opens a dialog listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

The number of model outputs must equal the number of plant unmeasured disturbance inputs. The white noise entering the model is assumed to have unity standard deviation.

Noise

Use these settings to model noise in the plant's measured outputs.

The example below shows the tab's appearance with the **Signal-by-signal** option selected for a plant having two measured outputs. The graphic shows the noise location.

Use the table to specify the character of each noise input.

The **Name** and **Units** columns are noneditable. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes there apply to the entire design.)

The **Type** column sets the noise character. To edit this, click the cell and select from the resulting menu. You have the following options:
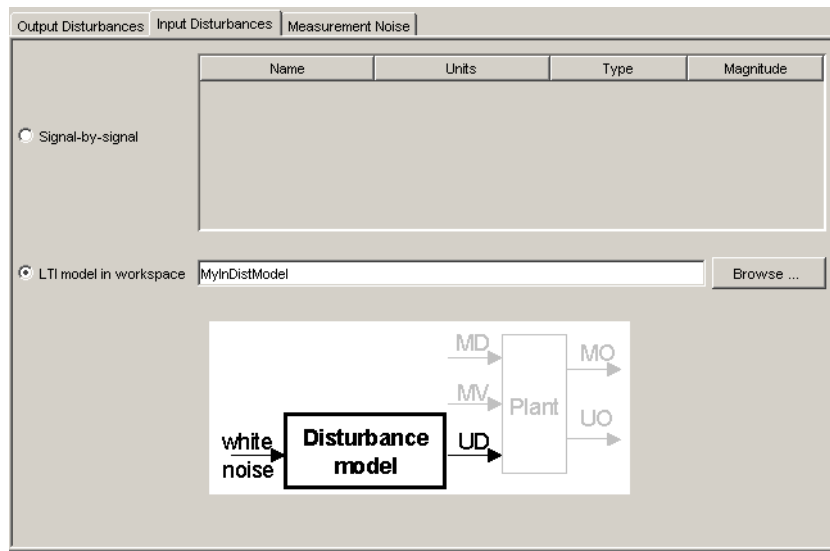
- **White** – white noise
- **Steps** – simulates random step-like disturbances (integrated white noise)

The **Magnitude** column specifies the standard deviation of the white noise assumed to create the noise. Set it to zero if you want to specify that an output is noise-free.

For example, if **Type** is **Steps** and **Magnitude** is 2, the noise model is integrated white noise, where the white noise has a standard deviation of 2.

If the above options are too restrictive, select the **LTI model in Workspace** option. The tab appearance changes as follows.
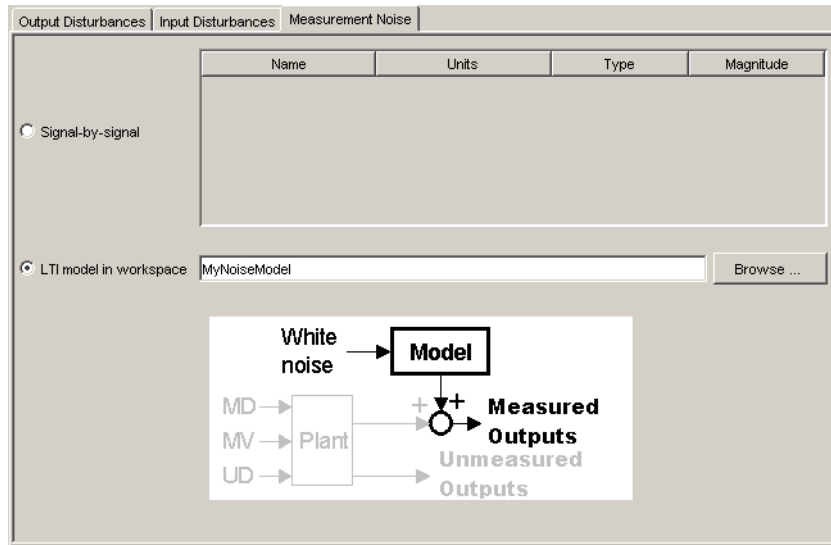
You must specify an LTI model residing in your workspace. The **Browse** button opens a dialog listing all LTI models in your workspace, and allows you to choose one. You can also type the model name in the edit box, as shown above.

The number of noise model outputs must equal the number of plant measured outputs.

The white noise entering the model is assumed to have unity standard deviation.

Right-Click Menus

Copy Controller
Creates a new controller having the same settings and a default name.

Delete Controller

Deletes the controller. If the controller is being used in a simulation scenario, the design tool replaces it with the first controller in your list, and displays a warning message.

Rename Controller

Opens a dialog allowing you to rename the controller.

Note Each controller in a design project/task must have a unique name.

Export Controller

Opens the MPC Exporter dialog (see "Exporting a Controller" on page 5-19).

Simulation Scenario View

This view appears whenever you select one of your scenario specification nodes (see "The Tree View" on page 5-7). It allows you to specify simulation settings and independent variables. All have default values, but you will want to change at least some of them (otherwise all independent variables will be constant). Defaults for a plant with three inputs and two outputs appears below.

The middle table won't appear unless you have designated at least one input signal to be a measured disturbance.

The following sections describe the view's main features.

• "Model and Horizons Tab" on page 5-37

Simulation Settings

Use this section to set the following:

- **Controller** – select one of your controllers
- **Plant** – select the plant model that will act as the "real" plant in the simulation, i.e., it need not be the same as that used for controller predictions.
- **Duration** – the simulation duration in time units
- **Close loops** – if unchecked, the simulation will be open-loop
- **Enforce Constraints** – if unchecked, all controller constraints will be ignored

The **Control interval** field is display-only, and reflects the setting in your **Controller** selection. You can change it there if necessary (see "Model and Horizons Tab" on page 5-37).

Setpoints

Note Setpoint specifications affect *closed-loop* simulations only.

Use this table to specify the setpoint for each output. In the example below, which is for an application having two plant outputs, the first would be constant at 0.0, and the second would change step-wise.

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes apply to the entire design.)

The **Type** column specifies the setpoint variation. To change this, click on the cell and select a choice from the resulting menu.

The significance of the **Initial value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see "Signal Type Settings" on page 5-64.

If the **Look ahead** option is checked (i.e., on), the controller will use future values of the setpoints in its calculations. This improves setpoint tracking, but knowledge of future setpoint changes is unusual in practice.

Note In the current implementation, checking or unchecking the **Look ahead** option for one output will set the others to the same state. The Model Predictive Control Toolbox code does not allow you to **Look ahead** for some outputs but not for others.

Measured Disturbances

Use this table to specify the variation of each measured disturbance. In the example below, which is for an application having a single measured disturbance, the "Steam Rate" input would be constant at 0.0.

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes apply to the entire design.)

The **Type** column specifies the disturbance variation. To change this, click on the cell and select a choice from the resulting menu.

The significance of the **Initial value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see "Signal Type Settings" on page 5-64.

If the **Look ahead** option is checked (i.e., on), the controller will use future values of the measured disturbance(s) in its calculations. This improves disturbance rejection, but knowledge of future disturbances is unusual in practice. *It has no effect in an open-loop simulation*.

Note In the current implementation, checking or unchecking the **Look ahead** option for one input will set the others to the same state. The Model Predictive Control Toolbox code does not allow you to **Look ahead** for some inputs but not for others.

Unmeasured Disturbances

Use this table to specify the variation of each measured unmeasured disturbance. In the example below, all would be constant at 0.0.

Unmeasured Disturbance Locations

You can simulate an unmeasured disturbance in any of the following locations:

- The plant's unmeasured disturbance (UD) inputs (if any)
- The plant's measured outputs (MO)
- The plant's manipulated variable (MV) inputs

All of the above will appear as rows in the table. In the case of a measured output or manipulated variable, the disturbance is an additive bias.

The **Name** and **Units** columns are display-only. To change them, use the signal definition view. (See "Signal Definition View" on page 5-21. Any changes apply to the entire design.)

The **Type** column specifies the disturbance variation. To change this, click on the cell and select a choice from the resulting menu.

The significance of the **Initial value**, **Size**, **Time**, and **Period** columns depends on the **Type**. If a cell is gray (noneditable), it doesn't apply to the **Type** you've chosen.

For details on the signal types, see "Signal Type Settings" on page 5-64.

Open-Loop Simulations

For open-loop simulations, you can vary the MV unmeasured disturbance to simulate the plant's response to a particular MV. The MV signal coming from the controller stays at its nominal value, and the MV unmeasured disturbance adds to it.

For example, suppose Reflux Rate is an MV, and the corresponding row in the table below represents an unmeasured disturbance in this MV.

Unmeasured disturbances

Name	Units	Type	Initial Value	Size	Time	Period
Feed Rate	kmol/min	Constant	0.0			
Distillate Purity	mol %	Constant	0.0			
Bottoms Purity	mol %	Constant	0.0			
Reflux Rate	kmol/min	Constant	0.0			

You could set it to a constant value of 1 to simulate the plant's open-loop unit-step response to the Reflux Rate input. (In a closed-loop simulation, controller adjustments would also contribute, changing the response.)

Similarly, an unmeasured disturbance in an MO adds to the output signal coming from the plant. If there are no changes at the plant input, the plant outputs are constant, and you see only the change due to the disturbance. This allows you to check the disturbance character before running a closed-loop simulation.

Signal Type Settings

The table below is an example that uses five of the six available signal types (the Constant option has been illustrated above). The cells with white backgrounds are the entries you must supply. All have defaults.

Name	Units	Type	Initial Value	Size	Time	Period
Distillate Purity	mol %	Step	0.0	1.0	1.0	
Bottoms Purity	mol %	Ramp	0.0	1.0	1.0	
Reflux Rate	kmol/min	Sine	0.0	1.0	0.0	1.0
Steam Rate	kmol/min	Pulse	0.0	1.0	0.0	1.0
Feed Rate	kmol/min	Gaussian	0.0	1.0	1.0	

Constant

The signal will be held at the specified **Initial value** for the entire simulation.

$$y = y_0 \text{ for } t \geq 0$$

Step

Prior to **Time**, the signal = **Initial value**. At **Time**, the signal changes step-wise by **Size** units. Its value thereafter = **Initial value** + **Size**.

$$y = y_0 \text{ for } 0 \leq t < t_0 \text{ where } y_0 = \textbf{Initial value}, t_0 = \textbf{Time}$$

$$y = y_0 + M \text{ for } t \geq t_0 \text{ where } M = \textbf{Size}$$

Ramp

Prior to **Time**, the signal = **Initial value**. At **Time**, the signal begins to vary linearly with slope **Size**.

$$y = y_0 \text{ for } 0 \leq t < t_0 \text{ where } y_0 = \textbf{Initial value}, t_0 = \textbf{Time}$$

$$y = y_0 + M(t - t_0) \text{ for } t \geq t_0 \text{ where } M = \textbf{Size}$$

Sine

Prior to **Time**, the signal = **Initial value**. At **Time**, the signal begins to vary sinusoidally with amplitude **Size** and period **Period**.

$y = y_0$ for $0 \leq t < t_0$ where y_0 = **Initial value**, t_0 = **Time**

$y = y_0 + M \sin[\omega(t - t_0)]$ for $t \geq t_0$ where M = **Size**, $\omega = 2\pi/$**Period**

Pulse

Prior to **Time**, the signal = **Initial value**. At **Time**, a square pulse of duration **Period** and magnitude **Size** occurs.

$y = y_0$ for $0 \leq t < t_0$ where y_0 = **Initial value**, t_0 = **Time**

$y = y_0 + M$ for $t_0 \leq t < t_0 + T$ where M = **Size**, T = **Period**

$y = y_0$ for $t \geq t_0 + T$

Gaussian

Prior to **Time**, the signal = **Initial value**. At **Time**, the signal begins to vary randomly about **Initial value** with standard deviation **Size**.

$y = y_0$ for $0 \leq t < t_0$ where y_0 = **Initial value**, t_0 = **Time**

$y = y_0 + M\text{randn}$ for $t \geq t_0$ where M = **Size**

randn is MATLAB's random-normal function, which generates random numbers having zero mean and unit variance.

Simulation Button

Click the **Simulate** button to simulate the scenario. You can also click **Ctrl-R**, use the toolbar icon (see "The Toolbar" on page 5-6), or use the **MPC/Simulate** menu option (see "Menu Bar" on page 5-3).

Right-Click Menus

Copy Scenario
Creates a new simulation scenario having the same settings and a default name.

Delete Scenario
Deletes the scenario.

Rename Scenario
Opens a dialog allowing you to rename the scenario.

Note Each scenario in a design project/task must have a unique name.

Response Plots

Each time you simulate a scenario, the design tool plots the corresponding plant input and output responses. The graphic below shows such a *response plot* for a plant having two outputs (the corresponding input response plot is not shown).
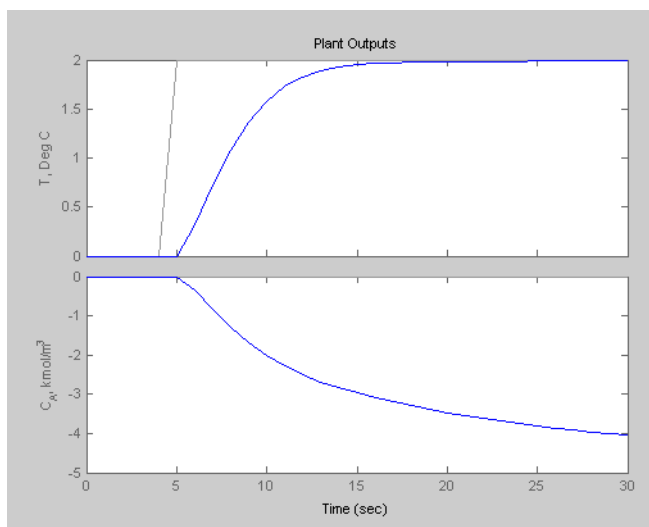
By default, each plant signal plots in its own graph area (as shown above). If the simulation is closed loop, each output signal plot include the corresponding setpoint.

The following sections describe response plot customization options.

Data Markers

You can use data markers to label a curve or to display numerical details.

Adding a Data Marker

To add a data marker, click on the desired curve at the location you want to mark. The following graph shows a marker added to each output response and its corresponding setpoint.
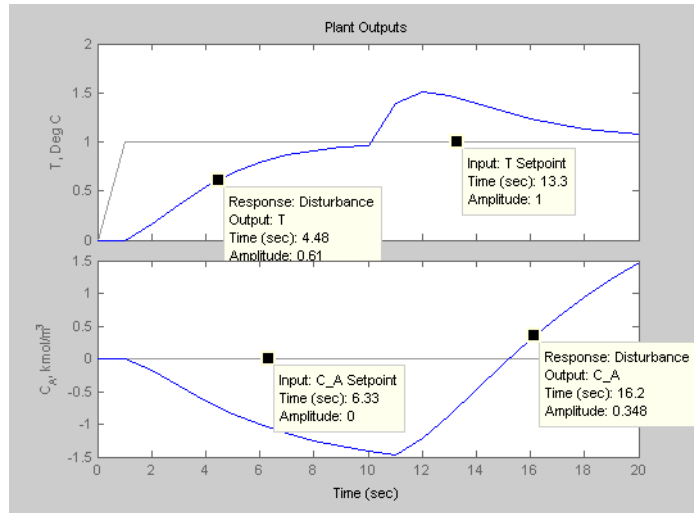
Data Marker Contents

Each data marker provides information about the selected point, as follows:

- **Response** – the *scenario* that generated the curve
- **Time** – the time value at the data marker location
- **Amplitude** – the signal value at the data marker location
- **Output** – the plant variable name (plant outputs only)
- **Input** – variable name for plant inputs and setpoints

Changing a Data Marker's Alignment

To relocate the data marker's label (without moving the marker), right-click on the marker, and select one of the four **Alignment** menu options. The above example shows three of the possible four alignment options.

Relocating a Data Marker

To move a marker, left-click on it (holding down the mouse key) and drag it along its curve to the desired location.

Deleting Data Markers

To delete all data markers in a plot, click in the plot's white space.

To delete a single data marker, right-click on it and select the **Delete** option.

Right-Click Options

Right-click on a data marker to use one of the following options:

- **Alignment** – relocate the marker's label
- **Font Size** – change the label's font size
- **Movable** – on/off option that makes the marker movable or fixed
- **Delete** – deletes the selected marker
- **Interpolation** – Interplolate linearly between the curve's data points, or locate at the nearest data point
- **Track Mode** – Changes the way the marker responds when you drag it

Displaying Multiple Scenarios

By default the response plots include all the scenarios you've simulated. The example below shows a response plot for a plant with two outputs. The data markers indicate the two scenarios being plotted: "Accurate Model" and "Perturbed Model". Both scenarios use the same setpoints (not marked – the lighter solid lines).

Viewing Selected Scenarios

If your plots are too cluttered, you can hide selected scenarios. To do so,

- Right-click in the plot's white space
- Choose **Responses** from the resulting context menu
- Toggle a response on or off using the submenu

Note This selection affects all variables being plotted.
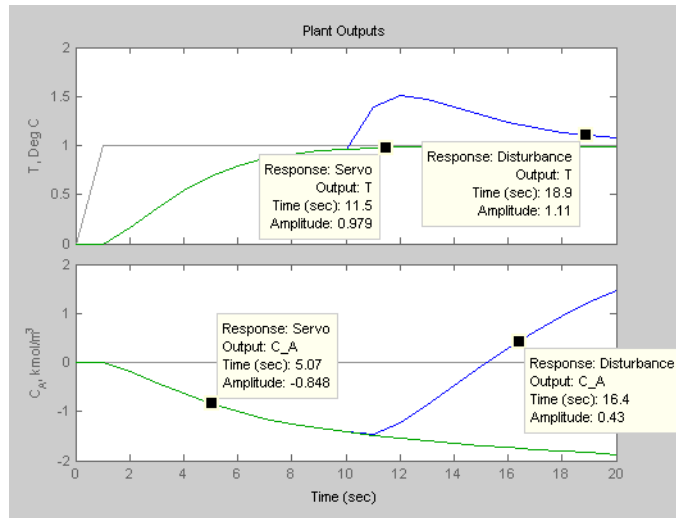
Revising a Scenario

If you modify and recalculate a scenario, its data are replotted, replacing the original curves.

Viewing Selected Variables

By default, the design tool plots all plant inputs in a single window, and plots all plant outputs in another. If your application involves many signals, the plots of each may be too small to view comfortably.

Therefore, you can control the variables being plotted. To do so, right-click in a plot's white space and choose **Channel Selector** from the resulting menu. A dialog box appears, on which you can opt to show or hide each variable.

Grouping Variables in a Single Plot

By default, each variable appears in its own plot area. You can instead choose to display variables together in a single plot. To do so, right-click in a plot's white space select **Channel Grouping**, and then select **All**.

To return to the default mode, use the **Channel Grouping**: **None** option.

Normalizing Response Amplitudes

When you're using the **Channel Grouping**: **All** option, you might find that the variables have very different scales, making it difficult to view them together. You can choose to *normalize* the curves, so that each expands or contracts to fill the available plot area.

For example, the plot below shows two plant outputs together (**Channel Grouping**: **All** option). The outputs have very different magnitudes. When plotted together, it's hard to see much detail in the smaller response.
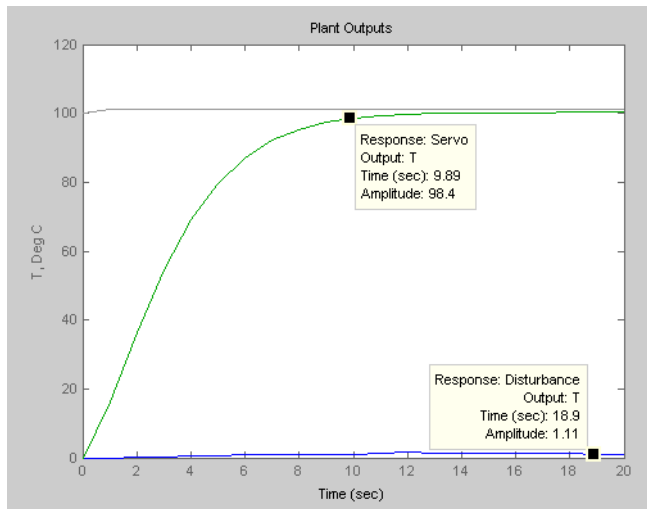
The plot below shows the normalized version, which displays each curve's variations clearly.

The y-axis scale is no longer meaningful, however. If you want to know a normalized signal's amplitude, use a data marker (see "Adding a Data Marker" on page 5-68). Note that the two data markers on the plot below are at the same normalized y-axis location, but correspond to very different amplitudes in the original (unnormalized) coordinates.

6
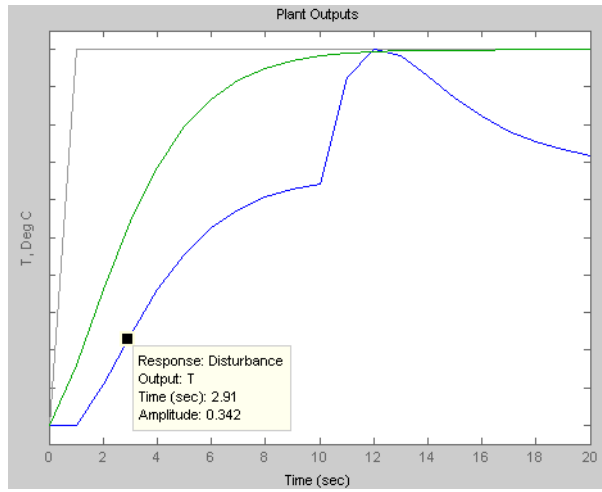
Function Reference

Functions — By Category

MPC Controller

Function Name	Description
d2d	Change MPC controller's sampling time
display	Display properties of MPC controller
get	Access/query property values
mpc	Create MPC controller
set	Set/modify MPC controller properties
setmpcsignals	Set signal types in MPC plant model
getname	Get I/O signal names in MPC prediction model
setname	Set I/O signal names in MPC prediction model
getmpcdata	Get privateMPC data structure
setmpcdata	Get privateMPC data structure

MPC Controller Characteristics

Function Name	Description
compare	Compare two MPC objects
isempty	Test true for empty MPC controller
mpcprops	Provide help on MPC controller's properties
mpchelp	MPC property and function help
mpcverbosity	Change the level of verbosity of the MPC Toolbox
pack	Reduce size of MPC object in memory
size	Display model output/input/disturbance dimensions

Linear Behavior of MPC Controller

Function Name	Description
cloffset	Compute MPC closed-loop DC gain from output disturbances to measured outputs assuming constraints are inactive at steady state
ss	Convert unconstrained MPC controller to state-space linear form
tf	Convert unconstrained MPC controller to linear transfer function
zpk	Convert unconstrained MPC controller to zero/pole/gain form

MPC State

Function Name	Description
mpcstate	Define state for MPC controller
trim	Compute the steady-state value of MPC controller state for given inputs and outputs values.
get	Access/query MPC state properties
set	Set/modify MPC state properties

MPC Computation and Simulation

Function Name	Description
mpcmove	Compute the MPC control action
sim	Simulate closed-loop/open-loop response to arbitrary reference and disturbance signals
mpcsimopt	Specify MPC simulation options
plot	Plot responses generated by MPC simulations

State Estimation

Function Name	Description
getestim	Extract model and gain used for observer design
setestim	Modify an MPC object's linear state estimator
getindist	Retrieves the unmeasured input disturbance model
setindist	Modify the unmeasured input disturbance model
getoutdist	Retrieve unmeasured output disturbance model
setoutdist	Modify the unmeasured output disturbance model

Quadratic Programming

Function Name	Description
qpdantz	Solve a convex quadratic program using Dantzig-Wolfe's algorithm
qpsolver	QP solver

Functions — Alphabetical List

This section contains function reference pages listed alphabetically.

cloffset

Purpose   Compute MPC closed-loop DC gain from output disturbances to measured outputs assuming constraints are inactive at steady state

Syntax   `DCgain=cloffset(MPCobj)`

Description   The `cloff` function computes the DC-gain from output disturbances to measured outputs, assuming constraints are not active, based on the feedback connection between `Model.Plant` and the linearized MPC controller, as depicted below.
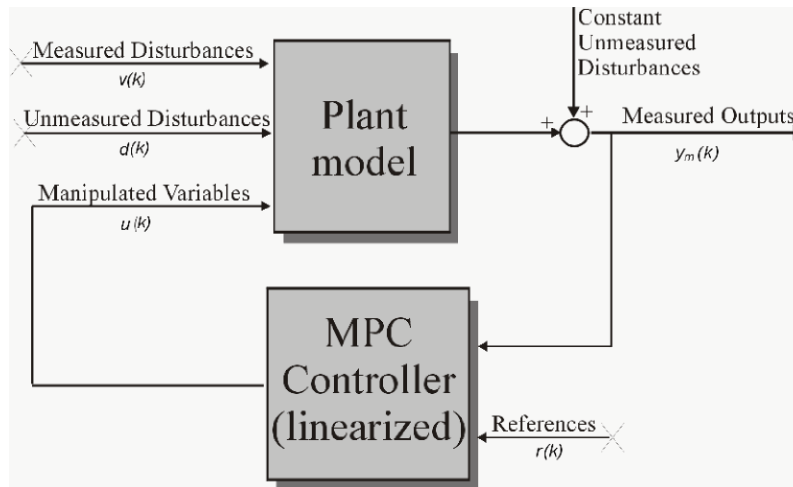
Computing the Effect of Output Disturbances

By superposition of effects, the gain is computed by zeroing references, measured disturbances, and unmeasured input disturbances.

`DCgain=cloffset(MPCobj)` returns an n_{ym}-by-n_{ym} DC gain matrix `DCgain`, where n_{ym} is the number of measured plant outputs. `MPCobj` is the MPC object specifying the controller for which the closed-loop gain is calculated. `DCgain(i,j)` represents the gain from an additive (constant) disturbance on output `j` to measured output `i`. If row `i` contains all zeros, there will be no steady-state offset on output `i`.

Examples   See `misocloffset.m` in `mpcdemos`.

See Also   `mpc`, `ss`

compare

Purpose   Compare two MPC objects

Syntax   yesno=compare(MPC1,MPC2)

Description   The compare function compares the contents of two MPC objects MPC1, MPC2. If the design specifications (models, weights, horizons, etc.) are identical, then yesno is equal to 1.

Note compare may return yesno=1 even if the two objects are not identical. For instance, MPC1 may have been initialized while MPC2 may have not, so that they may have different sizes in memory. In any case, if yesno=1 the behavior of the two controllers will be identical.

See Also   mpc, pack

Purpose   Change MPC controller's sampling time

Syntax   MPCobj=d2d(MPCobj,ts)

Description   The d2d function changes the sampling time of the MPC controller MPCobj to
                     ts. All models are sampled or resampled as soon as the QP matrices must be
                     computed, e.g., when sim or mpcmove are used.

See Also   mpc, set

get

Purpose	MPC property values

Syntax
```
Value = get(MPCobj,'PropertyName')
get(MPCobj)
Struct = get(MPCobj)
```

Description   `Value = get(MPCobj,'PropertyName')` returns the current value of the property PropertyName of the MPC controller MPCobj. The string `'PropertyName'` can be the full property name (for example, `'UserData'`) or any unambiguous case-insensitive abbreviation (for example, `'user'`). You can specify any generic MPC property.

`Struct = get(MPCobj)` converts the MPC controller MPCobj into a standard MATLAB structure with the property names as field names and the property values as field values.

`get(MPCobj)` without a left-side argument displays all properties of MPCobj and their values.

Remark   An alternative to the syntax

```
Value = get(MPCobj,'PropertyName')
```

is the structure-like referencing

```
Value = MPCobj.PropertyName
```

For example,

```
MPCobj.Ts
MPCobj.p
```

return the values of the sampling time and prediction horizon of the MPC controller MPCobj.

See Also   `mpc`, `set`

Purpose   Model and gain for observer design

Syntax   M=getestim(MPCobj)
                      [M,A,Cm]=getestim(MPCobj)
                      [M,A,Cm,Bu,Bv,Dvm]=getestim(MPCobj)
                      [M,model,Index]=getestim(MPCobj,'sys')

Description   M=getestim(MPCobj) extracts the estimator gain M used by the MPC controller
                      MPCobj for observer design. The observer is based on the models specified in
                      MPCobj.Model.Plant, in MPCobj.Model.Disturbance, by the output
                      disturbance model (default is integrated white noise, see "Output Disturbance
                      Model" on page 2-9), and by MPCobj.Model.Noise.

The state estimator is based on the linear model (see "State Estimation" on page 2-8)

$$x(k+1) = Ax(k) + B_u u(k) + B_v v(k)$$

$$y_m(k) = C_m x(k) + D_{vm} v(k)$$

where $v(k)$ are the measured disturbances, $u(k)$ are the manipulated plant inputs, $y_m(k)$ are the measured plant outputs, and $x(k)$ is the overall state vector collecting states of plant, unmeasured disturbance, and measurement noise models.

The estimator used in the MPC Toolbox is described in "State Estimation" on page 2-8. The estimator's equations are

Predicted Output Computation:

$$\hat{y}_m(k|k-1) = C_m \hat{x}(k|k-1) + D_{vm} v(k)$$

Measurement Update:

$$\hat{x}(k|k) = \hat{x}(k|k-1) + M(y_m(k) - \hat{y}_m(k|k-1))$$

Time Update:

$$\hat{x}(k+1|k) = A\hat{x}(k|k)) + B_u u(k) + B_v v(k)$$

By combining these three equations, the overall state observer is

$$\hat{x}(k+1|k) = (A - LC_m)\hat{x}(k|k) + Ly_m(k)B_u u(k) + (B_v - LD_{vm})v(k)$$

where $L=AM$.

`[M,A,Cm]=getestim(MPCobj)` also returns matrices A, C_m used for observer design. This includes plant model, disturbance model, noise model, offsets. The extended state is

x=plant states; disturbance models states; noise model states]

`[M,A,Cm,Bu,Bv,Dvm]=getestim(MPCobj)` retrieves the whole linear system used for observer design.

`[M,model,Index]=getestim(MPCobj,'sys')` retrieves the overall model used for observer design (specified in the `Model` field of the MPC object) as an LTI state-space object, and optionally a structure `Index` summarizing I/O signal types.

The extended input vector of model `model` is

u=[manipulated vars;measured disturbances; 1; noise exciting disturbance model;noise exciting noise model]

Model `model` has an extra measured disturbance input v=1 used for handling possible nonequilibrium nominal values (see "Offsets" on page 2-4).

Input, output, and state names and input/output groups are defined for model `model`.

The structure `Index` has the fields detailed in the following table.

Field Name	Description
ManipulatedVariables	Indices of manipulated variables within input vector
MeasuredDisturbances	Indices of measured disturbances within input vector (not including offset=1)
Offset	Index of offset=1

Field Name	Description
WhiteNoise	Indices of white noise signals within input vector
MeasuredOutputs	Indices of measured outputs within output vector
UnmeasuredOutputs	Indices of unmeasured outputs within output vector

The model returned by getestim does not include the additional white noise added on manipulated variables and measured disturbances to ease the solvability of the Kalman filter design, as described in Equation 2-5 on page 2-10.

See Also   setestim, mpc, mpcstate

getindist

Purpose   Unmeasured input disturbance model

Syntax   `model=getindist(MPCobj)`

Description   `model=getindist(MPCobj)` retrieves the linear discrete-time transfer function used to model unmeasured input disturbances in the MPC setup described by the MPC object `MPCobj`. Model `model` is an LTI object with as many outputs as the number of unmeasured input disturbances, and as many inputs as the number of white noise signals driving the input disturbance model.

See Figure 2-2, Model Used for State Estimation, on page 2-8 for details about the overall model used in the MPC algorithm for state estimation purposes.

See Also   `mpc`, `setindist`, `setestim`, `getestim`, `getoutdist`

Purpose   Private MPC data structure

Syntax   `mpcdata=getmpcdata(MPCobj)`

Description   `mpcdata=getmpcdata(MPCobj)` returns the private field `MPCData` of the MPC object `MPCobj`. Here, all internal QP matrices, models, estimator gains are stored at initalization of the object. You can manually change the private data structure using the `setmpcdata` command, although you may only need this for very advanced use of the MPC Toolbox.

Note Changes to the data structure may easily lead to unpredictable results.

See Also   `setmpcdata`, `set`, `get`

getname

Purpose   I/O signal names in MPC prediction model

Syntax   `name=getname(MPCobj,'input',I)`
                   `name=getname(MPCobj,'output',I)`

Description   `name=getname(MPCobj,'input',I)` returns the name of the I-th input signal
                   in variable name. This is equivalent to `name=MPCobj.Model.Plant.`
                   `InputName{I}`. The name property is equal to the contents of the corresponding
                   `Name` field of `MPCobj.DisturbanceVariables` or
                   `MPCobj.ManipulatedVariables`.

                   `name=getname(MPCobj,'output',I)` returns the name of the I-th output
                   signal in variable name. This is equivalent to
                   `name=MPCobj.Model.Plant.OutputName{I}`. The name property is equal to the
                   contents of the corresponding `Name` field of `MPCobj.OutputVariables`.

See Also   `setname`, `mpc`, `set`

getoutdist

Purpose   Unmeasured output disturbance model

Syntax   `outdist=getoutdist(MPCobj)`
                 `[outdist,channels]=getoutdist(MPCobj)`

Description `outdist=getoutdist(MPCobj)` retrieves the linear discrete-time transfer function used to model output disturbances in the MPC setup described by the MPC object `MPCobj`. Model `outdist` is an LTI object with as many outputs as the number of measured + unmeasured outputs, and as many inputs as the number of white noise signals driving the output disturbance model.

See Figure 2-2, Model Used for State Estimation, on page 2-8 for details about the overall model used in the MPC algorithm for state estimation purposes.

`[outdist,channels]=getoutdist(MPCobj)` also returns the output channels where integrated white noise was added as an output disturbance model. This is only meaningful when the default output disturbance model is used, namely when `MPCobj.OutputVariables(i).Integrators` is empty for all channels `i`. The array `channels` is empty for user-provided output disturbance models.

See Also   `mpc`, `setoutdist`, `setestim`, `getestim`, `getindist`

mpc

Purpose   Create MPC controller

Syntax   MPCobj=mpc(plant)
                   MPCobj=mpc(plant,ts)
                   MPCobj=mpc(plant,ts,p,m)
                   MPCobj=mpc(plant,ts,p,m,weights)
                   MPCobj=mpc(plant,ts,p,m,weights,MV,OV,DV)
                   MPCobj=mpc(models,ts,p,m,weights,MV,OV,DV)
                   MPCobj=mpc

Description   MPCobj=mpc(plant) creates a MPC controller based on the discrete-time model
                   model plant. The model can be specified either as an LTI object, or as an object
                   in the System Identification Toolbox's format (IDMODEL object), see Using
                   Identified Models (p. 2-19).

                   MPCobj=mpc(plant,ts) also specifies the sampling time ts for the MPC
                   controller. A continuous-time plant is discretized with sampling time ts. A
                   discrete-time plant is resampled if its sampling time is different than the
                   controller's sampling time ts. If plant is a discrete-time model with
                   unspecified sampling time, namely plant.ts=-1, then the MPC Toolbox
                   assumes that the plant is sampled with the controller's sampling time ts.

                   MPCobj=mpc(plant,ts,p,m) also specifies prediction horizon p and control
                   horizon m.

                   MPCobj=mpc(plant,ts,p,m,weights) also specifies the structure weights of
                   input, input increments, and output weights (see "Weights" on page 8-7).

                   MPCobj=mpc(plant,ts,p,m,weights,MV,OV,DV) also specifies limits on
                   manipulated variables (MV) and output variables (OV), as well as equal concern
                   relaxation values, units, etc. Names and units of input disturbances can be also
                   specified in the optional input DV. The fields of structures MV, OV, and DV are
                   described in "ManipulatedVariables" on page 8-3, in "OutputVariables" on
                   page 8-5, and in "DisturbanceVariables" on page 8-6, respectively).

                   MPCobj=mpc(models,ts,p,m,weights,MV,OV,DV) where model is a structure
                   containing models for plant, unmeasured disturbances, measured
                   disturbances, and nominal linearization values, as described in "Model" on
                   page 8-8.

                   MPCobj=mpc returns an empty MPC object.

Note Other MPC properties are specified by using set(MPCobj,Property1, Value1,Property2,Value2,...) or MPCobj.Property=Value.

Examples   Define an MPC controller based on the transfer function model $s+1/(s^2+2s)$, with sampling time T_s=0.1 s, and satisfying the input constraint -1≤ u ≤1:

```
Ts=.1;     %Sampling time
MV=struct('Min',-1,'Max',1);
p=20;
m=3;

mpc1=mpc(tf([1 1],[1 2 0]),Ts,p,m,[],MV);
```

See Also   set, get

mpchelp

Purpose   MPC property and function help

Syntax   mpchelp
                 mpchelp name
                 out=mpchelp(`name')
                 mpchelp(MPCobj
                 mpchelp(MPCobj,'name');
                 out=mpchelp(MPCobj,'name');

Description mpchelp provides a complete listing of Model Predictive Control Toolbox.

                mpchelp name provides on-line help for the function or property name.

                out=mpchelp(`name') returns the help text in string, out.

                mpchelp(obj) displays a complete listing of functions and properties for the
                MPC object, obj, along with the on-line help for the object's constructor.

                mpchelp(obj,'name') displays the help for function or property, name, for the
                MPC object, obj.

                out=mpchelp(obj,'name') returns the help text in string, out.

Examples   To get help on the MPC method "getoutdist", you can type

                   mpchelp getoutdist

See Also   mpcprops

Purpose   Compute MPC control action

Syntax   u=mpcmove(MPCobj,x,ym,r,v)
                    [u,Info]=mpcmove(MPCobj,x,ym,r,v)

Description   u=mpcmove(MPCobj,x,ym,r,v) computes the current input move $u(k)$, given the current estimated extended state $x(k)$, the vector of measured outputs $y_m(k)$, the reference vector $r(k)$, and the measured disturbance vector $v(k)$, by solving the quadratic programming problem based on the parameters contained in the MPC controller MPCobj.

x is an mpcstate object. It is updated by mpcmove through the internal state observer based on the extended prediction model (see getestim for details). A default initial state x for the first call at time k=0 can be simply defined as

    x=mpcstate(MPCobj)

[u,Info]=mpcmove(MPCobj,x,ym,r,v) also returns the structure Info containing details about the optimal control calculations. Info has the fields listed below.

Field Name	Description
Uopt	Optimal input trajectory over the prediction horizon, returned as a p-by-n_u dimensional array.
Yopt	Optimal output sequence over the prediction horizon, returned as a p-by-n_y dimensional array
Xopt	Optimal state sequence over the prediction horizon, returned as a p-by-n_x dimensional array, where n_x=total number of states of the extended state vector
Topt	Prediction time vector (0:p-1)'
Slack	Value of the ECR slack variable ε at optimum

Field Name	Description
Iterations	Number of iterations needed by the QP solver
QPCode	Exit code of the QP solver

To plot the optimal input trajectory, type

```
plot(Topt,Uopt)
```

The optimal output and state trajectories can be plotted similarly. The input, output, and state sequences Uopt, Yopt, Xopt, Topt correspond to the predicted open-loop optimal control trajectories solving the optimization problem described in "Optimization Problem" on page 2-5. The optimal trajectories might also help understand the closed-loop behavior. For instance, constraints that are active in the open-loop optimal trajectory only at late steps of the prediction horizon might not be active at all in the closed-loop MPC trajectories. The sequence of optimal manipulated variable increments can be retrieved from MPCobj.MPCData.MPCstruct.optimalseq.

QPCode returns either 'feasible', 'infeasible' or 'unreliable' (the latter occurs when the QP solver terminates because the maximum number of iterations MPCobj.Optimizer.MaxIter is exceeded; see qpdantz on page 6-33). When QPCode='infeasible', then u is obtained by shifting the previous optimal sequence of manipulated variable rates (stored in MPCobj.MPCData.MPCstruct.optimalseq inside the MPC object MPCobj), and summing the first entry of this sequence to the previous vector of manipulated moves. You may set up different backup strategies for handling infeasible situations by discarding u and replacing it with a different emergency decision-variable vector.

r/v can be either a sample (no future reference/disturbance known in advance) or a sequence of samples (when a preview / look-ahead / anticipative effect is desired). In the latter case, they must be an array with as many rows as p and as many columns as the number of outputs/measured disturbances, respectively. If the number of rows is smaller than p, the last sample is extended constantly over the horizon, to obtain the correct size.

The default for y and r is MPCobj.Model.Nominal.Y. The default for v is obtained from MPCobj.Model.Nominal.U. The default for x is

mpcstate(MPCobj,MPCobj.Model.Nominal.X,0,0,U0) where U0 are the entries from MPCobj.Model.Nominal.U corresponding to manipulated variables.

To bypass the MPC Controller block's internal estimator and use your own state observer to update the MPC state yourself, you can for instance use the syntax

```
xp=x.plant; xd=x.dist; xn=x.noise;     % Save current state
u=mpcmove(MPCobj,x,ym,r,v);            % x will be updated
% Now call to your state update function:
[xp,xd,xn]=my_estimator(xp,xd,xn,ym); % States get updated
x.plant=xp;x.dist=xd;x.noise=xn;
```

Examples   Model predictive control of a multi-input single-output system (see the demo MISO.M). The system has three inputs (one manipulated variable, one measured disturbance, one unmeasured disturbance) and one output.

```
% Open-loop system parameters

% True plant and true initial state
sys=ss(tf({1,1,1},{[1 .5 1],[1 1],[.7 .5 1]}));
x0=[0 0 0 0 0]';

% MPC object setup

Ts=.2;              % sampling time

% Define type of input signals
model.InputGroup=struct('Manipulated',1,'Measured',2,'Unmeasured
',3);

% Define constraints on manipulated variable
MV=struct('Min',0,'Max',1);

Model=[]; % Reset structure Model
Model.Plant=sys;
% Integrator driven by white noise with variance=1000
Model.Disturbance=tf(sqrt(1000),[1 0]);
```

```
p=[];      % Prediction horizon (take default one)
m=3;       % Control horizon
weights=[]; % Default value for weights

MPCobj=mpc(Model,Ts,p,m,weights,MV);

% Simulate closed loop system using MPCMOVE

Tstop=30;   %Simulation time

xmpc=mpcstate(MPCobj); % Initial state of MPC controller
x=x0;                  % Initial state of Plant
r=1;                   % Output reference trajectory

% State-space matrices of Plant model
[A,B,C,D]=ssdata(c2d(sys,Ts));

YY=[];XX=[];RR=[];
for t=0:round(Tstop/Ts)-1,
    XX=[XX,x];

    % Define measured disturbance signal
    v=0;
    if t*Ts>=10, v=1; end

    % Define unmeasured disturbance signal
    d=0;
    if t*Ts>=20, d=-0.5; end

    % Plant equations: output update
    % (note: no feedrthrough from MV to Y, D(:,1)=0)
    y=C*x+D(:,2)*v+D(:,3)*d;
    YY=[YY,y];

    % Compute MPC law
    u=mpcmove(MPCobj,xmpc,y,r,v);

    % Plant equations: state update
```

```
      x=A*x+B(:,1)*u+B(:,2)*v+B(:,3)*d;
   end

   % Plot results
   plot(O:Ts:Tstop-Ts,YY);grid
```

See Also   mpc, mpcstate, sim, setestim, getestim

mpcprops

Purpose　　　　Provide help on MPC controller's properties

Syntax　　　　`mpcprops`

Description　　`mpcprops` displays details on the generic properties of MPC controllers. It provides a complete list of all the fields of MPC objects with a brief description of each field and the corresponding default values.

See Also　　　`set`, `get`, `mpchelp`

Purpose   MPC simulation options

Syntax   SimOptions=mpcsimopt(mpcobj)

Description   The purpose of mpcsimopt is to create an object SimOptions of class @mpcsimopt for specifying additional parameters for simulation with sim.

SimOptions=mpcsimopt(mpcobj) creates an empty object SimOptions which is compatible with the MPC object mpcobj. The fields of the object SimOptions and their description are reported in Table 8-12, MPC Simulation Options Properties, on page 8-14.

Examples   We want to simulate the MPC control of a multi-input multi-output (MIMO) system under predicted / actual plant model mismatch (demo simmismatch.m). The system has two manipulated variables, two unmeasured disturbances, and two measured outputs.

```
% Open-loop system parameters
p1 = tf(1,[1 2 1])*[1 1; 0 1];
plant = ss([p1 p1]);

% Define I/O types
plant=setmpcsignals(plant,'MV',[1 2],'UD',[3 4]);

% Define I/O names (optional)
set(plant,'InputName',{'mv1','mv2','umd3','umd4'});

% Model for unmeasured input disturbances
distModel = eye(2,2)*ss(-.5,1,1,0);

% Create MPC object
mpcobj = mpc(plant,1,40,2);
mpcobj.Model.Disturbance = distModel;

% Closed-loop MPC simulation with model mismatch and unforeseen
% unmeasured disturbance inputs

% Define plant model generating the data
p2 = tf(1.5,[0.1 1 2 1])*[1 1; 0 1];
psim = ss([p2 p2 tf(1,[1 1])*[0;1]]);
```

```
psim=setmpcsignals(psim,'MV',[1 2],'UD',[3 4 5]);

% Closed-loop simulation
dist=ones(1,3); % Unmeasured disturbance trajectory
refs=[1 2];     % Output reference trajectory
Tf=100; % Total number of simulation steps

options=mpcsimopt(mpcobj);
options.unmeas=dist;
options.model=psim;

sim(mpcobj,Tf,refs,options);
```

See Also   sim

Purpose   Define MPC controller state

Syntax   xmpc=mpcstate(MPCobj,xp,xd,xn,u)
xmpc=mpcstate(MPCobj)

Description   xmpc=mpcstate(MPCobj,xp,xd,xn,u) defines an mpcstate object for state estimation and optimization in an MPC control algorithm based on the MPC object MPCobj. The state of an MPC controller contains the estimates of the states $x(k)$, $x_d(k)$, $x_m(k)$, where $x(k)$ is the state of the plant model, $x_d(k)$ is the overall state of the input and output disturbance model, $x_m(k)$ is the state of the measurement noise model, and the value of the last vector $u(k\text{-}1)$ of manipulated variables. The overall state is updated from the measured output $y_m(k)$ by a linear state observer (see "State Observer" on page 2-9).

xmpc=mpcstate(MPCobj) returns a default extended initial state that is compatible with the MPC controller MPCobj. Such a default state has plant state and previous input initialized at nominal values, and the states of the disturbance and noise models at zero.

Note that mpcstate objects are updated by mpcmove through the internal state observer based on the extended prediction model.

See Also   getoutdist, setoutdist, setindist, getestim, setestim, ss, mpcmove

mpcverbosity

Purpose   Change toolbox verbosity level

Syntax   mpcverbosity on
                   mpcverbosity off
                   mpcverbosity

Description   mpcverbosity on enables messages displaying default operations taken by the
                   MPC Toolbox during the creation and manipulation of MPC objects.

                   mpcverbosity off turns messages off.

                   mpcverbosity just shows the verbosity status.

                   By default, messages are turned on.

                   See also "Construction and Initialization" on page 8-12.

See Also   mpc

Purpose   Reduce size of MPC object in memory

Syntax   `pack(MPCobj)`

Description   `pack(MPCobj)` cleans up information build at initialization and stored in the `MPCData` field of the MPC object `MPCobj`. This reduces the amount of bytes in memory required to store the MPC object. For MPC objects based on large prediction models it is recommended to pack the object before saving the object to file, in order to minimize the size of the file.

See Also   `mpc`, `getmpcdata`, `setmpcdata`, `compare`

plot

Purpose   Plot responses generated by MPC simulations

Syntax   plot(MPCobj,t,y,r,u,v,d)

Description   plot(MPCobj,t,y,r,u,v,d) plots the results of a simulation based on the MPC
object MPCobj. t is a vector of length Nt of time values, y is a matrix of output
responses of size [Nt,Ny] where Ny is the number of outputs, r is a matrix of
setpoints and has the same size as y, u is a matrix of manipulated variable
inputs of size [Nt,Nu] where Nu is the number of manipulated variables, v is a
matrix of measured disturbance inputs of size [Nt,Nv] where Nv is the number
of measured disturbance inputs, and d is a matrix of unmeasured disturbance
inputs of size [Nt,Nd] where Nd is the number of unmeasured disturbances
input.

See Also   sim, mpc

Purpose   Solve convex quadratic program using Dantzig-Wolfe's algorithm

Syntax   ```
[xopt,lambda,how]=qpdantz(H,f,A,b,xmin)
[xopt,lambda,how]=qpdantz(H,f,A,b,xmin,maxiter)
```

**Description**      `[xopt,lambda,how]=qpdantz(H,f,A,b,xmin)` solves the convex quadratic program

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

subject to $Ax \le b, x \ge x_{min}$

using Dantzig-Wolfe's active set method [1]. The Hessian matrix `H` should be positive definite. By default, `xmin=1e-5`. Vector `xopt` is the optimizer. Vector `lambda` contains the optimal dual variables (Lagrange multipliers).

The exit flag `how` is either `'feasible'`, `'infeasible'` or `'unreliable'`. The latter occurs when the solver terminates because the maximum number `maxiter` of allowed iterations was exceeded.

The solver is implemented in `qpsolver.dll`. Dantzig-Wolfe's algorithm uses the direction of the largest gradient, and the optimum is usually found after about $n+q$ iterations, where $n=\dim(x)$ is the number of optimization variables, and $q=\dim(b)$ is the number of constraints. More than $3(n+q)$ iterations are rarely required (see Chapter 7.3 of [2]).

**Examples**         Solve a random QP problem using `quadprog` from the Optimization Toolbox and `qpdantz`.

```
n=50;      % Number of vars

H=rand(n,n);H=H'*H;H=(H+H')/2;
f=rand(n,1);
A=[eye(n);-eye(n)];
b=[rand(n,1);rand(n,1)];

x1=quadprog(H,f,A,b);
[x2,how]=qpdantz(H,f,A,b,-100*ones(n,1));
```

qpdantz

References

[1] Fletcher, R. *Practical Methods of Optimization*, John Wiley & Sons, Chichester, UK, 1987.

[2] Dantzig, G.B. *Linear Programming and Extensions*, Princeton University Press, Princeton, 1963.

Purpose   Set or modify MPC object properties

Syntax   ```
set(sys,'Property',Value)
set(sys,'Property1',Value1,'Property2',Value2,...)
set(sys,'Property')
set(sys)
```

Description   The set function is used to set or modify the properties of an MPC controller
(see "MPC Controller Object" on page 8-2 for background on MPC properties).
Like its Handle Graphics counterpart, set uses property name/property value
pairs to update property values.

set(MPCobj,'Property',Value) assigns the value Value to the property of the
MPC controller MPCobj specified by the string 'Property'. This string can be
the full property name (for example, 'UserData') or any unambiguous
case-insensitive abbreviation (for example, 'user').

set(MPCobj,'Property1',Value1,'Property2',Value2,...) sets multiple
property values with a single statement. Each property name/property value
pair updates one particular property.

set(MPCobj,'Property') displays admissible values for the property specified
by 'Property'. See "MPC Controller Object" on page 8-2 for an overview of
legitimate MPC property values.

set(sys) displays all assignable properties of sys and their admissible values.

See Also   mpc, get

setestim

Purpose Modify MPC object's linear state estimator

Syntax
```
setestim(MPCobj,M)
setestim(MPCobj,'default')
```

Description The setestim function modifies the linear estimator gain of an MPC object. The state estimator is based on the linear model (see "State Estimation" on page 2-8)

$$x(k+1) = Ax(k) + B_u u(k) + B_v v(k)$$

$$y_m(k) = C_m x(k) + D_{vm} v(k)$$

where $v(k)$ are the measured disturbances, $u(k)$ are the manipulated plant inputs, $y_m(k)$ are the measured plant outputs, and $x(k)$ is the overall state vector collecting states of plant, unmeasured disturbance, and measurement noise models. The order of the states in x is the following: plant states; disturbance models states; noise model states.

setestim(MPCobj,M), where MPCobj is an MPC object, changes the default Kalman estimator gain stored in MPCobj to that specified by matrix M.

setestim(MPCobj,'default') restores the default Kalman gain.

The estimator used in the MPC Toolbox is described in "State Estimation" on page 2-8. The estimator's equations are as follows.

Predicted Output Computation:
$$\hat{y}_m(k|k-1) = C_m \hat{x}(k|k-1) + D_{vm} v(k)$$

Measurement Update:
$$\hat{x}(k|k) = \hat{x}(k|k-1) + M(y_m(k) - \hat{y}_m(k|k-1))$$

Time Update:
$$\hat{x}(k+1|k) = A\hat{x}(k|k)) + B_u u(k) + B_v v(k)$$

By combining these three equations, the overall state observer is

$$\hat{x}(k+1|k) = (A - LC_m)\hat{x}(k|k) + Ly_m(k)B_u u(k) + (B_v - LD_{vm})v(k)$$

where *L=AM*.

Note The estimator gain *M* has the same meaning as the gain *M* in function DKALMAN of the Control System Toolbox.

Matrices A, B$_u$, B$_v$, C$_m$, D$_{vm}$ can be retrieved using getestim as follows:

```
[M,A,Cm,Bu,Bv,Dvm]=getestim(MPCobj)
```

As an alternative they can be retrieved from the internal structure MPCobj.MPCData.MPCstruct under the fields A,Bu,Bv,Cm,Dvm (see getmpcdata on page 6-15).

Examples To design an estimator by pole placement, you can use the commands

```
[M,A,Cm]=getestim(MPCobj);
L=place(A',Cm',observer_poles)';
M=A\L;
setestim(MPCobj,M);
```

assuming that the linear system *AM=L* is solvable.

Note The pair (A,C_m) describing the overall state-space realization of the combination of plant and disturbance models must be observable for the state estimation design to succeed. Observability is checked in the MPC Toolbox at two levels: (1) observability of the plant model is checked *at construction* of the MPC object, provided that the model of the plant is given in state-space form; (2) observability of the overall extended model is checked *at initialization* of the MPC object, after all models have been converted to discrete-time, delay-free, state-space form and combined together (see the note on page 2-11).

---

setestim

See Also   getestim, mpc, mpcstate

Purpose   Modify unmeasured input disturbance model

Syntax   setindist(MPCobj,'integrators')
                 setindist(MPCobj,'model,model)

Description setindist(MPCobj,'integrators') imposes the default disturbance model
                for unmeasured inputs, that is for each unmeasured input disturbance
                channel, an integrator is added unless there is a violation of observability,
                otherwise the input is treated as white noise with unit variance (this is
                equivalent to MPCobj.Model.Disturbance=[]).

                setindist(MPCobj,'model',model) sets the input disturbance model to model
                (this is equivalent to MPCobj.Model.Disturbance=model).

See Also   mpc, getindist, setestim, getestim, setoutdist

setmpcdata

Purpose   Set private MPC data structure

Syntax   setmpcdata(MPCobj,mpcdata)

Description   setmpcdata(MPCobj,mpcdata) changes the private field MPCData of the MPC object MPCobj, where all internal QP matrices, models, estimator gains are stored at initalization of the object. You may only need this for very advanced use of the MPC Toolbox.

Note Changes to the data structure may easily lead to unpredictable results.

See Also   getmpcdata, set, get, pack

Purpose   Set signal types in MPC plant model

Syntax   `P=setmpcsignals(P,SignalType1,Channels1,SignalType2,Channels2,...)`

Description   The purpose of `setmpcsignals` is to set I/O channels of the MPC plant model `P`.
`P` must be an LTI object. Valid signal types, their abbreviations, and the
channel type they refer to are listed below.

Signal Type	Abbreviation	Channel
Manipulated	MV	Input
MeasuredDisturbances	MD	Input
UnmeasuredDisturbances	UD	Input
MeasuredOutputs	MO	Output
UnmeasuredOutputs	UO	Output

Unambiguous abbreviations of signal types are also accepted.

`P=setmpcsignals(P)` sets channel assignments to default, namely all inputs
are manipulated variables (MVs), all outputs are measured outputs (MOs).
More generally, input signals that are not explicitly assigned are assumed to
be MVs, while unassigned output signals are considered as MOs.

Examples   We want to define an MPC object based on the LTI discrete-time plant model
`sys` with four inputs and three outputs. The first and second input are
measured disturbances, the third input is an unmeasured disturbance, the
fourth input is a manipulated variable (default), the second output is an
unmeasured, all other outputs are measured

```
sys=setmpcsignals(sys,'MD',[1 2],'UD',[3],'UO',[2]);
mpc1=mpc(sys);
```

setmpcsignals

Note When using `setmpcsignals` to modify an existing MPC object, be sure that the fields `Weights`, `MV`, `OV`, `DV`, `Model.Noise`, and `Model.Disturbance` are consistent with the new I/O signal types.

See Also   `mpc`, `set`

Purpose   Set I/O signal names in MPC prediction model

Syntax   setname(MPCobj,'input',I,name)
                 setname(MPCobj,'output',I,name)

Description setname(MPCobj,'input',I,name) changes the name of the I-th input signal
                 to name. This is equivalent to MPCobj.Model.Plant.InputName{I}=name. Note
                 that setname also updates the read-only Name fields of
                 MPCobj.DisturbanceVariables and MPCobj.ManipulatedVariables.

                 setname(MPCobj,'output',I,name) changes the name of the I-th output
                 signal to name. This is equivalent to MPCobj.Model.Plant.OutputName{I}
                 =name. Note that setname also updates the read-only Name field of
                 MPCobj.OutputVariables.

                 _____

                 Note The Name properties of ManipulatedVariables, OutputVariables, and
                 DisturbanceVariables are read-only. You must use setname to assign signal
                 names, or equivalently modify the Model.Plant.InputName and
                 Model.Plant.OutputName properties of the MPC object.

                 _____

See Also   getname, mpc, set

setoutdist

Purpose   Modify unmeasured output disturbance model

Syntax   setoutdist(MPCobj,'integrators')
setoutdist(MPCobj,'remove',channels)
setoutdist(MPCobj,'model',model)

Description   setoutdist(MPCobj,'integrators') specifies the default method output
disturbance model, based on the specs stored in
MPCobj.OutputVariables.Integrator and
MPCobj.Weights.OutputVariables. Output integrators are added according to
the following rule:

1 Outputs are ordered by decreasing output weight (in case of time-varying
weights, the sum of the absolute values over time is considered for each
output channel. In case of equal output weight, the order within the output
vector is followed);

2 By following such order, an output integrator is added per measured
outputs, unless there is a violation of observability or the corresponding
value in MPCobj.OutputVariables.Integrator is zero. A warning message
is given when an integrator is added on an unmeasured output channel.

setoutdist(MPCobj,'remove',channels) removes integrators from the
output channels specified in vector channels. This corresponds to setting
MPCobj.OutputVariables(channels).Integrator=0. The default for
channels is (1:ny), where ny is the total number of outputs, that is, all output
integrators are removed.

setoutdist(MPCobj,'model',model) replaces the array of output integrators
designed by default according to MPCobj.OutputVariables.Integrator with
the LTI model model. The model must have ny outputs. If no model is specified,
then the default model based on the specs stored in
MPCobj.OutputVariables.Integrator and
MPCobj.Weights.OutputVariables is used (same as setoutdist(MPCobj,
'integrators').

See Also   mpc, getestim, setestim, setoutdist, setindist

Purpose   Simulate closed-loop/open-loop response to arbitrary reference and
disturbance signals

Syntax   sim(MPCobj,T,r)
sim(MPCobj,T,r,v)
sim(MPCobj,T,r,SimOptions) or sim(MPCobj,T,r,v,SimOptions)
[y,t,u,xp,xmpc,SimOptions]=sim(MPCobj,T,...)

Description   The purpose of sim is to simulate the MPC controller in closed-loop with a
linear time-invariant model, which, by default, is the plant model contained in
MPCobj.Model.Plant. As an alternative sim can simulate the open-loop
behavior of the model of the plant, or the closed-loop behavior in the presence
of a model mismatch between the prediction plant model and the model of the
process generating the output data.

sim(MPCobj,T,r) simulates the closed-loop system formed by the plant model
specified in MPCobj.Model.Plant and by the MPC controller specified by the
MPC object MPCobj, and plots the simulation results. T is the number of
simulation steps. r is the reference signal array with as many columns as the
number of output variables.

sim(MPCobj,T,r,v)   also specifies the measured disturbance signal v, that
has as many columns as the number of measured disturbances.

Note The last sample of r/v is extended constantly over the simulation
horizon, to obtain the correct size.

---

sim(MPCobj,T,r,SimOptions) or sim(MPCobj,T,r,v,SimOptions) specifies
the simulation options object SimOptions, such as initial states, input/output
noise and unmeasured disturbances, plant mismatch, etc. See mpcsimopt for
details.

Without output arguments, sim automatically plots input and output
trajectories.

[y,t,u,xp,xmpc,SimOptions]=sim(MPCobj,T,...) instead of plotting
closed-loop trajectories returns the sequence of plant outputs y, the time
sequence t (equally spaced by MPCobj.Ts), the sequence u of manipulated

variables generated by the MPC controller, the sequence xp of states of the model of the plant used for simulation, the sequence xmpc of states of the MPC controller (provided by the state observer), and the options object SimOptions used for the simulation.

The descriptions of the input arguments and their default values are shown in the table below.

Input Argument	Description	Default
MPCobj	MPC object specifying the parameters of the MPC control law	None
T	Number of simulation steps	Largest row-size of r,v,d,n
r	Reference signal	MPCobj.Model.Nominal.Y
v	Measured disturbance signal	Entries from MPCobj.Model.Nominal.U
SimOptions	Object of class @mpcsimopt containing the simulation parameters (See mpcsimopt)	[]

r is an array with as many columns as outputs, v is an array with as many columns as measured disturbances. The last sample of r/v/d/n is extended constantly over the horizon, to obtain the correct size.

The output arguments of sim are detailed below.

Output Argument	Description
y	Sequence of controlled plant outputs (without noise added on measured ones)
t	Time sequence (equally spaced by MPCobj.Ts)

Output Argument	Description
u	Sequence of manipulated variables generated by MPC
xp	Sequence of states of plant model (from Model or SimOptions.Model)
xmpc	Sequence of states of MPC controller (estimates of the extended state) This is a structure with the same fields as the mpcstate object.

Examples

We want to simulate the MPC control of a multi-input single-output system (the same model as in demo misosim.m). The system has one manipulated variable, one measured disturbance, one unmeasured disturbance, and one output.

```
%Plant model and initial state
sys=ss(tf({1,1,1},{[1 .5 1],[1 1],[.7 .5 1]}));

% MPC object setup
Ts=.2;              % sampling time
sysd=c2d(sys,Ts); % prediction model

% Define type of input signals
sysd=setmpcsignals(model,'MV',1,'MD',2,'UD',3);

MPCobj=mpc(sysd); % Default weights and horizons

% Define constraints on manipulated variable
MPCobj.MV=struct('Min',0,'Max',1);

Tstop=30;          % Simulation time

Tf=round(Tstop/Ts); % Number of simulation steps
r=ones(Tf,1);       % Reference trajectory
v=[zeros(Tf/3,1);ones(2*Tf/3,1)]; % Measured dist. trajectory
sim(MPCobj,Tf,r,v);
```

See Also   mpcsimopt, mpc, mpcmove

size

Purpose   Display model output/input/disturbance dimensions

Syntax   sizes=size(MPCobj)

Description sizes=size(MPCobj) returns the row vector sizes = $[n_{ym} \; n_u \; n_{yu} \; n_v \; n_d]$ associated with the MPC object MPCobj, where n_{ym} is the number of measured controlled outputs, n_u is the number of manipulated inputs, n_{yu} is the number of unmeasured controlled outputs, n_v is the number of measured disturbances, and n_d is the number of unmeasured disturbances.

size(MPCobj) by itself makes a nice display.

See Also   mpc, set

Purpose	Convert unconstrained MPC controller to state-space linear form

Syntax

```
sys=ss(MPCobj)
[sys,Br,Dr,Bv,Dv,Boff,Doff,But,Dut]=ss(MPCobj)
[sys,Br,Dr,Bv,Dv,Boff,Doff,But,Dut]=ss(MPCobj,ref_preview,
    md_preview,name_flag)
```

Description

The `ss` utility returns the linear controller sys as an LTI system in `ss` form corresponding to the MPC controller `MPCobj` when the constraints are not active. The purpose is to use the linear equivalent control in the Control System Toolbox for sensitivity analysis and other linear analysis.

`sys=ss(MPCobj)` returns the linear discrete-time dynamic controller sys

$$x(k+1) = Ax(k) + By_m(k)$$

$$u(k) = Cx(k) + Dy_m(k)$$

where y_m is the vector of measured outputs of the plant, and u is the vector of manipulated variables. The sampling time of controller sys is `MPCobj.Ts`.

`[sys,Br,Dr,Bv,Dv,Boff,Doff,But,Dut]=ss(MPCobj)` returns the linearized MPC controller in its full version, that has the following structure

$$x(k+1) = Ax(k) + By_m(k) + B_r r(k) + B_v v(k) + B_{ut} u_{\text{target}}(k) + B_{\text{off}}$$

$$u(k) = Cx(k) + Dy_m(k) + D_r r(k) + D_v v(k) + D_{ut} u_{\text{target}}(k) + D_{\text{off}}$$

Note vector x includes the states of the observer (plant+disturbance+noise model states) and the previous manipulated variable $u(k$-1).

In the general case of nonzero offsets, y_m (as well as r, v, u_{target}) must be interpreted as the difference between the vector and the corresponding offset. Vectors B_{off}, D_{off} are constant terms due to nonzero offsets, in particular they are nonzero if and only if `MPCobj.Model.Nominal.DX` is nonzero (continuous-time prediction models), or `MPCobj.Model.Nominal.Dx-MPCobj.Model.Nominal.X` is nonzero

(discrete-time prediction models). Note that when `Nominal.X` is an equilibrium state, B_{off}, D_{off} are zero.

Only the following fields of `MPCobj` are used when computing the state-space model: `Model`, `PredictionHorizon`, `ControlHorizon`, `Ts`, `Weights`.

`[sys,...]=ss(MPCobj,ref_preview,md_preview,name_flag)` allows you to specify if the MPC controller has preview actions on the reference and measured disturbance signals. If the flag `ref_preview='on'`, then matrices B_r and D_r multiply the whole reference sequence:

$$x(k+1) = Ax(k) + By_m(k) + B_r[r(k);r(k+1);...;r(k+p-1)] + ...$$

$$u(k) = Cx(k) + Dy_m(k) + D_r[r(k);r(k+1);...;r(k+p-1)] + ...$$

Similarly if the flag `md_preview='on'`, then matrices B_r and D_r multiply the whole measured disturbance sequence:

$$x(k+1) = Ax(k) + ... + B_v[v(k);v(k+1);...;v(k+p)] + ...$$

$$u(k) = Cx(k) + ... + D_v[v(k);v(k+1);...;v(k+p)] + ...$$

The optional input argument `name_flag='names'` adds state, input, and output names to the created LTI object.

Examples   To get the transfer function LTIcon from (y_m,r) to u,

```
[sys,Br,Dr]=ss(MPCobj);
set(sys,'B',[sys.B,Br],'D',[sys.D,Dr]);
```

See Also   `mpc`, `set`, `tf`, `zpk`

Purpose   Convert unconstrained MPC controller to linear transfer function

Syntax   `sys=tf(MPCobj)`

Description   The `tf` function computes the transfer function of the linear controller `ss(MPCobj)` as an LTI system in `tf` form corresponding to the MPC controller when the constraints are not active. The purpose is to use the linear equivalent control in the Control System Toolbox for sensitivity and other linear analysis.

See Also   `ss, zpk`

trim

Purpose   Compute steady-state value of MPC controller state for given inputs and outputs

Syntax   x=trim(MPCobj,y,u)

Description   The trim function finds a steady-state value for the plant state vector such that $x=Ax+Bu$, $y=Cx+Du$, or the best approximation of such an x in a least squares sense, sets noise and disturbance model states at zero, and forms the extended state vector.

See Also   mpc, mpcstate

Purpose   Convert unconstrained MPC controller to zero/pole/gain form

Syntax   `sys=zpk(MPCobj)`

Description   The `zpk` function computes the zero-pole-gain form of the linear controller
                   `ss(MPCobj)` as an LTI system in `zpk` form corresponding to the MPC controller
                   when the constraints are not active. The purpose is to use the linear equivalent
                   control in the Control System Toolbox for sensitivity and other linear analysis.

See Also   `ss`, `tf`

zpk

Block Reference

Blocks — Alphabetical List (p. 7-2)   Description of available block in the MPC Toolbox

7

Blocks — Alphabetical List

This section contains function reference pages listed alphabetically.

Purpose   Compute the MPC control law

Library   MPC Simulink Library
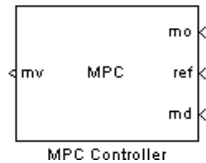
Description   The MPC Controller Block receives the current measured output, reference signal, and measured disturbance signal, and outputs the optimal manipulated variables by solving a quadratic program. The block is based on an MPC object, which provides performance and constraint specifications, as well as the sampling time of the block.

MPC Controller

Dialog Box
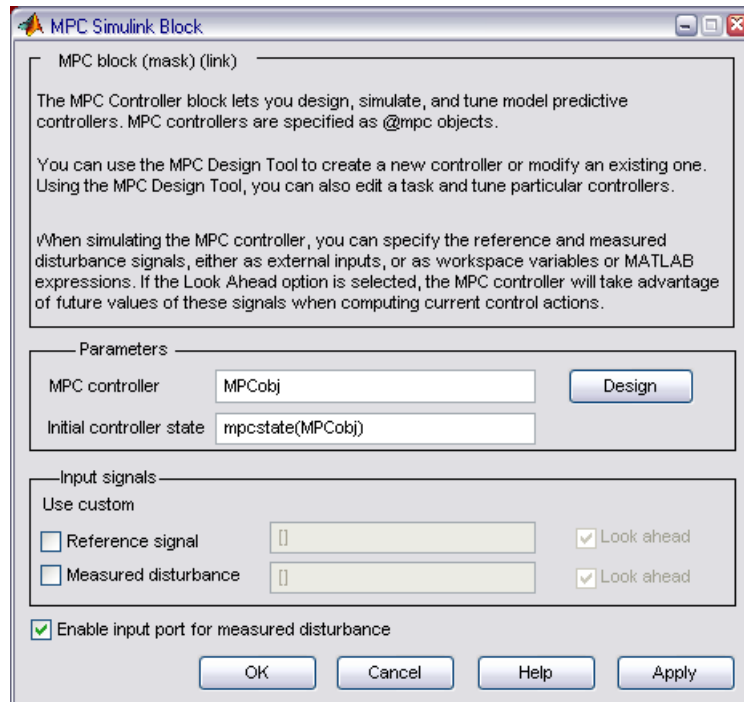
MPC controller

MPC controller object, defining prediction model, weights, constraints, sampling time, etc.

Initial controller state

Initial state of the MPC controller. This must be a valid `mpcstate` object.

Reference signal

Choose whether to load the reference signal from the workspace or from the block input port. If the signal is received from the workspace, the **Look Ahead** check box is enabled, and you can select the anticipative action (see "Look Ahead and Signals from the Workspace" on page 3-4).

Measured disturbance

Choose whether to load the disturbance signal from the workspace or from the block input port. If the signal is received from the workspace, the **Look Ahead** check box is enabled, and you can select the anticipative action (see "Look Ahead and Signals from the Workspace" on page 3-4).

The mask requires that you specify a valid MPC controller object. There are two ways of providing an MPC controller object:

• Load an existing MPC object from the workspace.

• Click the **Design** button to open the MPC Design Tool and design the MPC controller object there. You can run closed-loop simulations while the you are editing the MPC controller in the MPC Design Tool. In this case, the controller parameters used for simulating the Simulink diagram are those specified in the MPC Design Tool, so that you can easily tune the parameters of the controller.

In the latter case, you must export the designed controller as an MPC object to the workspace when the MPC Design Tool is closed, so that you can use the new changes for simulation.

When no MPC controller is specified in the edit box, clicking the **Design** button constructs a default MPC controller by obtaining a linearized model from the Simulink diagram.

Use the **Enable input port for measured disturbance** check box to resize the block. If the check box is not selected, the block has two input signals, namely measured outputs and references. When the check box is selected, the block has measured disturbances as the third input signal.

The **Enable input port for externally supplied manipulated variables to plant** check box is useful when you need to switch between MPC control and

another type of control (e.g., manual control) during a simulation. When the additional input port is enabled, the block is resized and a new input signal to the MPC controller appears, which represents the actual manipulated variables that the process is receiving. The actual inputs to the plant are used by the MPC algorithm to update the internal state estimate, rather than the manipulated variables generated by the MPC controller. The following example shows possible connections.
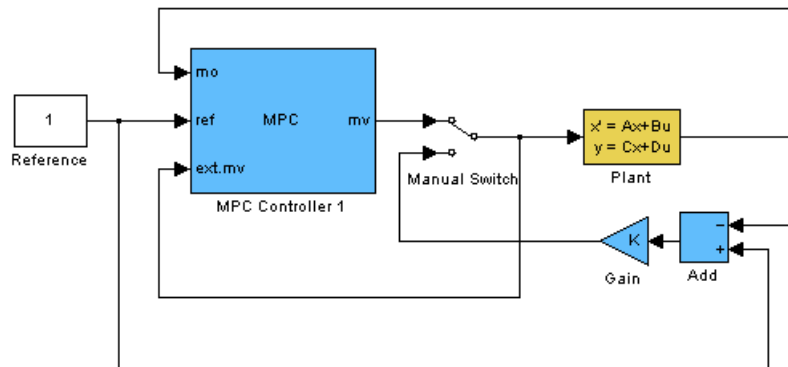
Bumpless Switching Between MPC and Another Controller

If the input port is disabled, or it is enabled but no signal is connected to it, then the MPC controller updates the internal state estimate by using the MPC action evaluated at the current time instant.

Note The MPC Controller block is a discrete-time block with sampling time inherited by the MPC object. The MPC block has direct feedthrough from measured outputs (*mo*), output references (*ref*), and measured disturbances (*md*) to MPC-manipulated variables (*mv*), and no direct feedthrough from externally supplied manipulated variables (*ext.mv*) to MPC-manipulated variables (*mv*).

See Also   mpc, mpcstate

MPC Controller Block

Object Reference

MPC Controller Object

All the parameters defining the MPC control law (prediction horizon, weights, constraints, etc.) are stored in an MPC object, whose properties are listed in Table 8-1.

Table 8-1: MPC Controller Object

Property	Description
`ManipulatedVariables` (or MV or `Manipulated` or `Input`)	Input and input-rate upper and lower bounds, ECR values, names, units, and input target
`OutputVariables` (or OV or `Controlled` or `Output`)	Output upper and lower bounds, ECR values, names, units
`DisturbanceVariables` (or DV or `Disturbance`)	Disturbance names and units
`Weights`	Weights defining the performance function
`Model`	Plant, input disturbance, and output noise models, and nominal conditions.
`Ts`	Controller's sampling time
`Optimizer`	Parameters for the QP solver
`PredictionHorizon`	Prediction horizon
`ControlHorizon`	Number of free control moves or vector of blocking moves
`History`	Creation time
`Notes`	User notes (text)
`UserData`	Any additional data

Table 8-1: MPC Controller Object (Continued)

Property	Description
MPCData (private)	Matrices for the QP problem and other accessorial data
Version (private)	MPC Toolbox version number

ManipulatedVariables

ManipulatedVariables (or MV or Manipulated or Input) is an n_u-dimensional array of structures (n_u = number of manipulated variables), one per manipulated variable. Each structure has the fields described in Table 8-2, where p denotes the prediction horizon.

Table 8-2: Structure ManipulatedVariables

Field Name	Content	Default
Min	1 to p dimensional vector of lower constraints on a manipulated variable u	-Inf
Max	1 to p dimensional vector of upper constraints on a manipulated variable u	Inf
MinECR	1 to p dimensional vector describing the equal concern for the relaxation of the lower constraints on u	0
MaxECR	1 to p dimensional vector describing the equal concern for the relaxation of the upper constraints on u	0
Target	1 to p dimensional vector of target values for the input variable u	0
RateMin	1 to p dimensional vector of lower constraints on the rate of a manipulated variable u	-Inf if problem is unconstrained, otherwise -10

Table 8-2: Structure ManipulatedVariables (Continued)

Field Name	Content	Default
RateMax	1 to p dimensional vector of upper constraints on the rate of a manipulated variable u	Inf
RateMinECR	1 to p dimensional vector describing the equal concern for the relaxation of the lower constraints on the rate of u	0
RateMaxECR	1 to p dimensional vector describing the equal concern for the relaxation of the upper constraints on the rate of u	0
Name	Name of input signal. This is inherited from InputName of the LTI plant model	InputName of LTI plant model
Units	String specifying the measurement units for the manipulated variable	''

Note Rates refer to the difference $\Delta u(k)=u(k)-u(k-1)$. Constraints and weights based on derivatives du/dt of continuous-time input signals must be properly reformulated for the discrete-time difference $\Delta u(k)$, using the approximation $du/dt \cong \Delta u(k)/T_s$.

OutputVariables

OutputVariables (or OV or Controlled or Output) is an n_y-dimensional array of structures (n_y = number of outputs), one per output signal. Each structure has the fields described in Table 8-3, where p denotes the prediction horizon.

Table 8-3: Structure OutputVariables

Field Name	Content	Default
Min	1 to p dimensional vector of lower constraints on an output y	-Inf
Max	1 to p dimensional vector of upper constraints on an output y	Inf
MinECR	1 to p dimensional vector describing the equal concern for the relaxation of the lower constraints on an output y	1
MaxECR	1 to p dimensional vector describing the equal concern for the relaxation of the upper constraints on an output y	1
Name	Name of output signal. This is inherited from OutputName of the LTI plant model	OutputName of LTI plant model
Units	String specifying the measurement units for the measured output	''
Integrator	Magnitude of integrated white noise on the output channel (0=no integrator)	[]

In order to reject constant disturbances due for instance to gain nonlinearities, the default output disturbance model used in the MPC Toolbox is a collection of integrators driven by white noise on measured outputs (see "Output Disturbance Model" on page 2-9). Output integrators are added according to the following rule:

1 Measured outputs are ordered by decreasing output weight (in case of time-varying weights, the sum of the absolute values over time is considered

for each output channel, and in case of equal output weight, the order within the output vector is followed).

2 By following such order, an output integrator is added per measured outputs, unless there is a violation of observability, or the user forces it by zeroing the corresponding value in OutputVariables.Integrators).

By default, OutputVariables.Integrators is empty on all outputs. This enforces the default action of the MPC Toolbox, namely add integrators on measured outputs, do not add integrators on unmeasured outputs. By setting the entry of OutputVariables(i).Integrators to zero, no attempt will be made to add integrated white noise on the i-th output . On the contrary, by setting the entry of OutputVariables(i).Integrators to one, an attempt will be made to add integrated white noise on the i-th output (see getoutdist on page 6-17).

DisturbanceVariables

DisturbanceVariables (or DV or Disturbance) is an $(n_v + n_d)$-dimensional array of structures (n_v = number of measured input disturbances, n_d = number of unmeasured input disturbances), one per input disturbance. Each structure has the fields described in Table 8-4.

Table 8-4: Structure DisturbanceVariables

Field Name	Content	Default
Name	Name of input signal. This is inherited from InputName of the LTI plant model	InputName of LTI plant model
Units	String specifying the measurement units for the manipulated variable	''

The order of the disturbance signals within the array DisturbanceVariables is the following: the first n_v entries relate to measured input disturbances, the last n_d entries relate to unmeasured input disturbances.

> **Note** The `Name` properties of `ManipulatedVariables`, `OutputVariables`, and `DisturbanceVariables` are read only. You can set signal names in the `Model.Plant.InputName` and `Model.Plant.OutputName` properties of the MPC object, for instance by using the method `setname`.

Weights

`Weights` is the structure defining the weighting matrices. Unlike the `InputSpecs` and `OutputSpecs` that are arrays of structures, `weights` is a single structure, whose fields are described in Table 8-5, where p denotes the prediction horizon, n_u the number of manipulated variables, n_y the number of output variables.

Table 8-5: Structure Weights

Field Name	Content	Default
`ManipulatedVariables` (or `MV` or `Manipulated` or `Input`)	n_u-by-(1 to p) dimensional array of input weights	`zeros(nu,1)`
`ManipulatedVariablesRate` (or `MVRate` or `ManipulatedRate` or `InputRate`)	n_u-by-(1 to p) dimensional array of input-rate weights	`0.1*ones(nu,1)`
`OutputVariables` (or `OV` or `Controlled` or `Output`)	n_y-by-(1 to p) dimensional array of output weights	1 (The default for output weights is the following: if $n_u>=n_y$, all outputs are weighted with unit weight; if $n_u<n_y$, n_u outputs are weighted with unit weight (with a preference on measured outputs), while the remaining outputs are not weighted.)
`ECR`	Weight on the slack variable ε used for softening the constraints	1e5*(max weight)

`ManipulatedVariables`, `ManipulatedVariablesRate`, and `OutputVariables` are arrays with as many columns as inputs or outputs. If weights are time invariant, then `ManipulatedVariables`, `ManipulatedVariablesRate`, and `OutputVariables` are row vectors, while for time-varying weights, they are an array with up to p rows. The last row is repeated by default in case the matrix has a number of rows smaller than the prediction horizon p.

The default ECR weight is 10^5 times the largest weight specified in `ManipulatedVariables`, `ManipulatedVariablesRate`, and `OutputVariables`.

Note All weights must be greater than or equal to zero. If all weights on manipulated variable increments are strictly positive, the resulting QP problem is always strictly convex. If some of those weights are zero, the Hessian matrix of the QP problem may become only positive semidefinite. In order to keep the QP problem always strictly convex, if the condition number of the Hessian matrix $K_{\Delta U}$ is larger than 10^{12}, the quantity `10*sqrt(eps)` is added on each diagonal term. This may only occur when all input rates are not weighted ($W^{\Delta u}=0$) (see "Cost Function" on page 2-15).

Model

The property `Model` specifies plant, input disturbance, and output noise models, and nominal conditions, according to the model setup described in Figure 2-2. It is specified through a structure containing the fields reported in Table 8-6.

Table 8-6: Structure Model Describing the Models Used by MPC

Field Name	Content	Default
`Plant`	LTI model (or IDMODEL) of the plant	No default
`Disturbance`	LTI model describing color of input disturbances	An integrator on each `Unmeasured` input channel

Table 8-6: Structure Model Describing the Models Used by MPC (Continued)

Field Name	Content	Default
Noise	LTI model describing color of plant output measurement noise	Unit white noise on each measured output = identity static gain
Nominal	Structure containing the state, input, and output values where Model.Plant is linearized	See Table 8-9.

Note Direct feedthrough from manipulated variables to measured outputs in Model.Plant is not allowed. See "Prediction Model" on page 2-2.

The type of input and output signals is assigned either through the InputGroup and OutputGroup properties of Model.Plant, or, more conveniently, through function setmpcsignals, according to the nomenclature described in Table 8-7 and Table 8-8.

Table 8-7: Input Groups in Plant Model

Name	Value
ManipulatedVariables (or MV or Manipulated or Input)	Indices of manipulated variables
MeasuredDisturbances (or MD or Measured)	Indices of measured disturbances
UnmeasuredDisturbances (or UD or Unmeasured)	Indices of unmeasured disturbances

Table 8-8: Output Groups in Plant Model

Name	Value
MeasuredOutputs (or MO or Measured)	Indices of measured outputs
UnmeasuredOutputs (or UO or Unmeasured)	Indices of unmeasured outputs

By default, all inputs are manipulated variables, and all outputs are measured.

Note With this current release, the InputGroup and OutputGroup properties of LTI objects are defined as structures, rather than cell arrays (see the Control System Toolbox documentation for more details).

The structure Nominal contains the nominal values for states, inputs, outputs and state derivatives/differences at the operating point where Model.Plant was linearized. The fields are reported in Table 8-9 (see "Offsets" on page 2-4).

Table 8-9: Nominal Values at Operating Point

Field	Description	Default
X	Plant state at operating point	0
U	Plant input at operating point, including manipulated variables, measured and unmeasured disturbances	0
Y	Plant output at operating point	0
DX	For continuous-time models, DX is the state derivative at operating point: DX=f(X,U). For discrete-time models, DX=$x(k+1)$-$x(k)$=f(X,U)-X.	0

Ts

Sampling time of the MPC controller. By default, if `Model.Plant` is a discrete-time model, `Ts=Model.Plant.ts`. For continuous-time plant models, you must specify a sampling time for the MPC controller.

Optimizer

Parameters for the QP optimization. Optimizer is a structure with the fields reported in Table 8-10.

Table 8-10: Optimizer Properties

Field	Description	Default
MaxIter	Maximum number of iterations allowed in the QP solver	200
Trace	On/off	'off'
Solver	QP solver used (only 'ActiveSet')	'ActiveSet'
MinOutputECR	Minimum positive value allowed for OutputMinECR and OutputMaxECR	1e-10

`MinOutputECR` is a positive scalar used to specify the minimum allowed ECR for output constraints. If values smaller than `MinOutputECR` are provided in the `OutputVariables` property of the MPC objects a warning message is issued and the value is raised to `MinOutputECR`.

PredictionHorizon

`PredictionHorizon` is an integer value expressing the number p of sampling steps of prediction.

ControlHorizon

`ControlHorizon` is either a number of free control moves, or a vector of blocking moves (see "Optimization Variables" on page 2-14).

History

`History` stores the time the MPC controller was created.

Notes

Notes stores user's notes as a cell array of strings.

UserData

Any additional data stored within the MPC controller object

MPCData

MPCData is a private property of the MPC object used for storing intermediate operations, QP matrices, internal flags, etc. See getmpcdata on page 6-15 and setmpcdata on page 6-40.

Version

Version is a private property indicating the MPC Toolbox version number.

Construction and Initialization

An MPC object is built in two steps. The first step happens *at construction* of the object when the object constructor mpc is invoked, or properties are changed by a set command. At this first stage, only basic consistency checks are performed, such as dimensions of signals, weights, constraints, etc. The second step happens *at initialization* of the object, namely when the object is used for the first time by methods such as mpcmove and sim, that require the full computation of the QP matrices and the estimator gain. At this second stage, further checks are performed, such as a test of observability of the overall extended model.

Informative messages are displayed in the command window in both phases, you can turn them on or off using the mpcverbosity command.

MPC State Object

The mpcstate object type contains the state of an MPC controller. Its properties are listed in Table 8-11.

Table 8-11: MPC State Object Properties

Property	Description
Plant	Array of plant states. Values are absolute, i.e., they include possible state offsets (cf. Model.Nominal.X)
Disturbance	Array of states of unmeasured disturbance models. This contains the states of the input disturbance model and, appended below, the states of the unmeasured output disturbances model
Noise	Array of states of measurement noise model
LastInput	Array of previous manipulated variables $u(k$-1). Values are absolute, i.e., they include possible input offsets (cf. Model.Nominal.U).

The command

```
mpcstate(mpcobj)
```

returns a zero extended initial state compatible with the MPC object mpcobj, and with mpcobj.Plant and mpcobj.LastInput initialized at the nominal values specified in mpcobj.Model.Nominal.

MPC Simulation Options Object

The mpcsimopt object type contains various simulation options for simulating an MPC controller with the command sim. Its properties are listed in Table 8-12.

Table 8-12: MPC Simulation Options Properties

Property	Description
PlantInitialState	Initial state vector of plant model generating the data
ControllerInitialState	Initial condition of the MPC controller. This must be a valid @mpcstate object
UnmeasuredDisturbance	Unmeasured disturbance signal entering the plant
InputNoise	Noise on manipulated variables
OutputNoise	Noise on measured outputs
RefLookAhead	Preview on reference signal ('on' or 'off')
MDLookAhead	Preview on measured disturbance signal ('on' or 'off')
Constraints	Use MPC constraints ('on' or 'off')
Model	Model used in simulation for generating the data.
StatusBar	Display wait bar ('on' or 'off')
MVSignal	Sequence of manipulated variables (with offsets) for open-loop simulation (no MPC action)
OpenLoop	Performs open-loop simulation

The command

```
SimOptions=mpcsimopt(mpcobj)
```

returns an empty `@mpcsimopt` object. You must use `set` / `get` to change simulation options.

`UnmeasuredDisturbance` is an array with as many columns as unmeasured disturbances, `InputNoise` and `MVSignal` are arrays with as many columns as manipulated variables, `OutputNoise` is an array with as many columns as measured outputs. The last sample of the array is extended constantly over the horizon to obtain the correct size.

Note Nonzero values of `ControllerInitialState.LastMove` are only meaningful if there are constraints on the increments of the manipulated variables.

The property `Model` is useful for simulating the MPC controller under model mismatch. The LTI object specified in `Model` can be either a replacement for `Model.Plant`, or a structure with fields `Plant`, `Nominal`. By default, `Model` is equal to `MPCobj.Model` (no model mismatch). If `Model` is specified, then `PlantInitialState` refers to the initial state of `Model.Plant` and is defaulted to `Model.Nominal.x`.

If `Model.Nominal` is empty, `Model.Nominal.U` and `Model.Nominal.Y` are inherited from `MPCobj.Model.Nominal`. `Model.Nominal.X/DX` is only inherited if both plants are state-space objects with the same state dimension.

Index