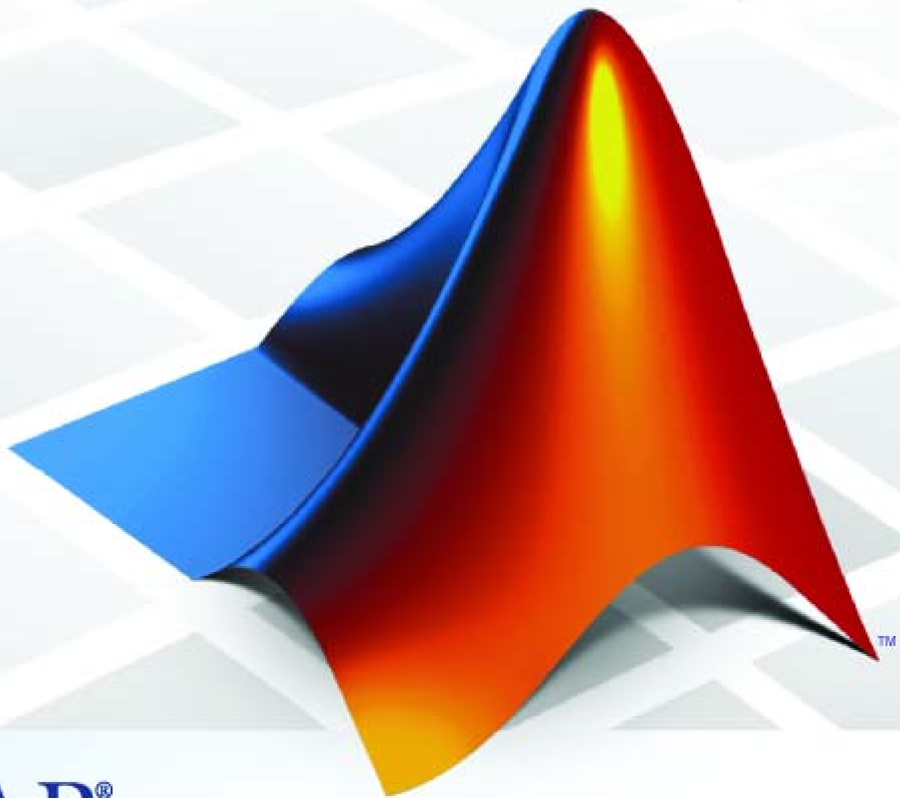
Model Predictive Control Toolbox™ 3
Reference

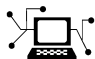
Alberto Bemporad
Manfred Morari
N. Lawrence Ricker



MATLAB®

The MathWorks™
Accelerating the pace of engineering and science

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 site.

Model Predictive Control Toolbox™ Reference

© COPYRIGHT 2005–2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

October 2004	First printing	New for Version 2.1 (Release 14SP1)
March 2005	Online only	Revised for Version 2.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 2.2.2 (Release 2006a)
September 2006	Online only	Revised for Version 2.2.3 (Release 2006b)
March 2007	Online only	Revised for Version 2.2.4 (Release 2007a)
September 2007	Online only	Revised for Version 2.3 (Release 2007b)
March 2008	Online only	Revised for Version 2.3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0 (Release 2008b)
March 2009	Online only	Revised for Version 3.1 (Release 2009a)
September 2009	Online only	Revised for Version 3.1.1 (Release 2009b)
March 2010	Online only	Revised for Version 3.2 (Release 2010a)

Contents

1 Function Reference

General 1-2

Creating MPC Controllers 1-2

Data Extraction 1-3

Conversions 1-3

Analysis 1-4

Controller Design 1-4

QP Solver 1-4

Simulink 1-5

2 Functions – Alphabetical List

3 Block Reference

4 Object Reference

MPC Controller Object **4-2**
ManipulatedVariables **4-3**
OutputVariables **4-4**
DisturbanceVariables **4-6**
Weights **4-6**
Model **4-9**
Ts **4-11**
Optimizer **4-11**
PredictionHorizon **4-12**
ControlHorizon **4-12**
History **4-12**
Notes **4-12**
UserData **4-12**
MPCData **4-13**
Version **4-13**
Construction and Initialization **4-13**

MPC Simulation Options Object **4-14**

MPC State Object **4-16**

Index

1

Function Reference

General (p. 1-2)
Creating MPC Controllers (p. 1-2)
Data Extraction (p. 1-3)
Conversions (p. 1-3)
Analysis (p. 1-4)
Controller Design (p. 1-4)
QP Solver (p. 1-4)
Simulink (p. 1-5)

General

`mpchelp`	MPC property and function help
`mpcprops`	Provide help on MPC controller's properties
`mpcverbosity`	Change toolbox verbosity level

Creating MPC Controllers

`gpc2mpc`	Generate MPC controller using generalized predictive controller (GPC) settings
`mpc`	Create MPC controller
`mpcstate`	Define MPC controller state
`set`	Set or modify MPC object properties
`setestim`	Modify MPC object's linear state estimator
`setindist`	Modify unmeasured input disturbance model
`setmpcdata`	Set private MPC data structure
`setmpcsignals`	Set signal types in MPC plant model
`setname`	Set I/O signal names in MPC prediction model
`setoutdist`	Modify unmeasured output disturbance model

Data Extraction

`get`	MPC property values
`getestim`	Model and gain for observer design
`getindist`	Unmeasured input disturbance model
`getmpcdata`	Private MPC data structure
`getname`	I/O signal names in MPC prediction model
`getoutdist`	Unmeasured output disturbance model
`sim`	Simulate closed-loop/open-loop response to arbitrary reference and disturbance signals

Conversions

`d2d`	Change MPC controller's sampling time
`pack`	Reduce size of MPC object in memory
`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

Analysis

`cloffset`	Compute MPC closed-loop DC gain from output disturbances to measured outputs assuming constraints are inactive at steady state
`compare`	Compare two MPC objects
`mpcmove`	Compute MPC control action
`mpcsimopt`	MPC simulation options
`plot`	Plot responses generated by MPC simulations
`sensitivity`	Compute effect of controller tuning weights on performance
`sim`	Simulate closed-loop/open-loop response to arbitrary reference and disturbance signals
`trim`	Compute steady-state value of MPC controller state for given inputs and outputs

Controller Design

`mpctool`	Start Model Predictive Controller GUI

QP Solver

`qpdantz`	Solve convex quadratic program using Dantzig-Wolfe's algorithm

Simulink

mpclib	MPC block library browser

2

Functions – Alphabetical List

cloffset

Purpose

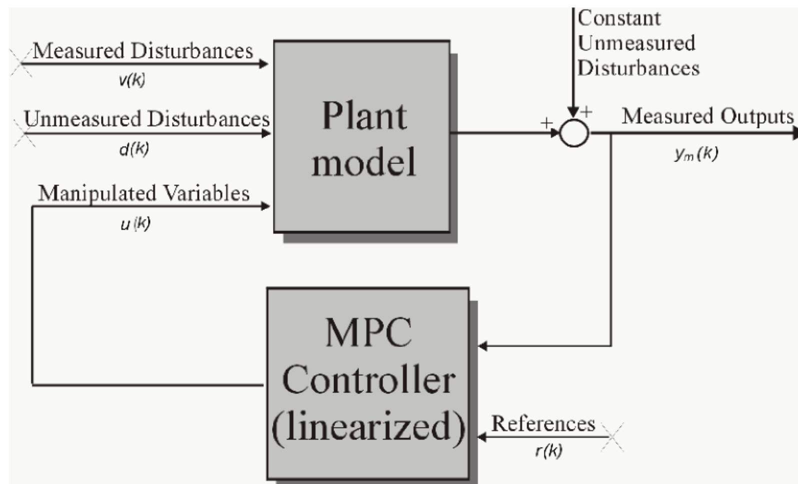
Compute MPC closed-loop DC gain from output disturbances to measured outputs assuming constraints are inactive at steady state

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`

Purpose

MPC property values

Syntax

```
Value = get(MPCobj,'PropertyName')
Struct = get(MPCobj)
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" in the Model Predictive Control Toolbox™ User's Guide), `and by` `MPCobj.Model.Noise`.

The state estimator is based on the linear model (see "State Estimation" in the Model Predictive Control Toolbox User's Guide)

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

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

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 Model Predictive Control Toolbox software is described in "State Estimation". 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:

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

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

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

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" in the Model Predictive Control Toolbox User's Guide)

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
`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 (2–6) the Model Predictive Control Toolbox User's Guide.

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 “Model Used for State Estimation” in the Model Predictive Control Toolbox User’s Guide 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 Model Predictive Control Toolbox software.

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`

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 "Model Used for State Estimation" in the Model Predictive Control Toolbox User's Guide 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 meaninful 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`

Purpose

Generate MPC controller using generalized predictive controller (GPC) settings

Syntax

```
mpc = gpc2mpc(plant)
gpcOptions = gpc2mpc
mpc = gpc2mpc(plant,gpcOptions)
```

Description

`mpc = gpc2mpc(plant)` generates a single-input single-output MPC controller with default GPC settings and sampling time of the plant, *`plant`*. The GPC is a nonminimal state-space representation described in [1]. *`plant`* is a discrete-time LTI model with sampling time greater than 0.

`gpcOptions = gpc2mpc` creates a structure *`gpcOptions`* containing default values of GPC settings.

`mpc = gpc2mpc(plant,gpcOptions)` generates an MPC controller using the GPC settings in *`gpcOptions`*.

Tips

- For plants with multiple inputs, only one input is the manipulated variable, and the remaining inputs are measured disturbances in feedforward compensation. The plant output is the measured output of the MPC controller.
- Use the MPC controller with Model Predictive Control Toolbox software for simulation and analysis of the closed-loop performance.

Input Arguments

`plant`

Discrete-time LTI model with sampling time greater than 0.

`gpcOptions`

GPC settings, specified as a structure with the following fields.

`N1`	Starting interval in prediction horizon, specified as a positive integer. **Default:** 1.
`N2`	Last interval in prediction horizon, specified as a positive integer greater than `N1`. **Default:** 10.
`NU`	Control horizon, specified as a positive integer less than the prediction horizon. **Default:** 1.
`Lam`	Penalty weight on changes in manipulated variable, specified as a positive integer greater than or equal to 0. **Default:** 0.
`T`	Numerator of the GPC disturbance model, specified as a row vector of polynomial coefficients whose roots lie within the unit circle. **Default:** [1].
`MVindex`	Index of the manipulated variable for multi-input plants, specified as a positive integer. **Default:** 1.

Examples

Design an MPC controller using GPC settings:

```
% Specify the plant described in Example 1.8  of [1].
G = tf(9.8*[1 -0.5 6.3],conv([1 0.6565],[1 -0.2366 0.1493]));

% Discretize the plant with sample time of 0.6 seconds.
Ts = 0.6;
Gd = c2d(G, Ts);
```

```
% Create a GPC settings structure.
GPCoptions = gpc2mpc;

% Specify the GPC settings described in example 4.11 of [1].
% Hu
GPCoptions.NU = 2;
% Hp
GPCoptions.N2 = 5;
% R
GPCoptions.Lam = 0;
GPCoptions.T = [1 -0.8];

% Convert GPC to an MPC controller.
mpc = gpc2mpc(Gd, GPCoptions);

% Simulate for 50 steps with unmeasured disturbance between
% steps 26 and 28, and reference signal of 0.
SimOptions = mpcsimopt(mpc);
SimOptions.UnmeasuredDisturbance = [zeros(25,1); ...
-0.1*ones(3,1); 0];
sim(mpc, 50, 0, SimOptions);
```

References

[1] Maciejowski, J. M. *Predictive Control with Constraints*, Pearson Education Ltd., 2002, pp. 133–142.

See Also

"MPC Controller Object" on page 4-2

How To

- "Model Predictive Control Problem Setup"
- "Designing Controllers Using the Design Tool GUI"
- "Designing Controllers Using the Command Line"

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 an MPC controller based on the discrete-time model `plant`. The model can be specified either as an LTI object, or as an object in System Identification Toolbox™ format (IDMODEL object). See "Using Identified Models" in the Model Predictive Control Toolbox User's Guide.

`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 Model Predictive Control Toolbox software 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 4-6).

`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 4-3, in "OutputVariables" on page 4-4, and in "DisturbanceVariables" on page 4-6, 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 4-9.

`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`

Purpose

MPC property and function help

Syntax

```
mpchelp
mpchelp name
out=mpchelp(`name')
mpchelp(obj)
mpchelp(obj,'name')
out=mpchelp(obj,'name')
```

Description

`mpchelp` provides a complete listing of Model Predictive Control Toolbox help.

`mpchelp name` provides online 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 online 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`

mpcmove

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 following fields.

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" in the Model Predictive Control Toolbox User's Guide. 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`). 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 MPC Control of a Multi-Input Single-Output System). 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
sys.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(0:Ts:Tstop-Ts,YY);grid
```

See Also

mpc, mpcstate, sim, setestim, getestim

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 MPC Simulation Options Properties on page 4-14.

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 $ym(k)$ by a linear state observer (see “State Observer” in the Model Predictive Control Toolbox User’s Guide).

`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`

Purpose

Start Model Predictive Controller GUI

Syntax

```
mpctool
mpctool(MPCobj)
mpctool(MPCobj,'objname')
mpctool(MPCobj1, MPCobj2, ...)
mpctool(MPCobj1, 'objname1', MPCobj2, 'objname2', ...)
mpctool('TaskName')
```

Description

`mpctool` starts the GUI. For more information about designing and testing model predictive controllers, see "Reference for the Design Tool GUI" in the Model Predictive Control Toolbox User's Guide.

`mpctool(MPCobj)` starts the GUI and loads `MPCobj`, which is an existing controller object.

`mpctool(MPCobj,'objname')` assigns objname (specified as a string) to the controller you are loading into the GUI. If you do not specify a name, the GUI uses the name of the variable that stores the controller object.

`mpctool(MPCobj1, MPCobj2, ...)` loads the specified list of controllers.

`mpctool(MPCobj1, 'objname1', MPCobj2, 'objname2', ...)` loads the specified list of controllers and assigns each controller the specified name.

`mpctool('TaskName')` starts the GUI and creates a new Model Predictive Control design task with the name specified by the string `'TaskName'`.

See Also

`mpc`

mpcverbosity

Purpose Change toolbox verbosity level

Syntax

```
mpcverbosity on
mpcverbosity off
mpcverbosity
```

Description `mpcverbosity on` enables messages displaying default operations taken by Model Predictive Control Toolbox software during the creation and manipulation of model predictive control 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 4-13 .

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

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

$$\text{subject to } Ax \le b, x \ge x_{min}$$

using Dantzig-Wolfe's active set method [2]. 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.mex`. 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™ software 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));
```

Bibliography

[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

Compute effect of controller tuning weights on performance

Syntax

```
[J, sens] = sensitivity(MPCobj, PerfFunc, PerfWeights, Tstop,
   r, v, simopt, utarget)
[J, sens] = sensitivity(MPCobj,'perf_fun',param1,param2,...)
```

Description

The `sensitivity` function is a controller tuning aid. J specifies a scalar performance metric. `sensitivity` computes J and its partial derivatives with respect to the controller tuning weights. These *sensitivities* suggest tuning weight adjustments that should improve performance, i.e., reduce J.

`[J, sens] = sensitivity(MPCobj, PerfFunc, PerfWeights, Tstop, r, v, simopt, utarget)` calculates the scalar performance metric, `J`, and sensitivities, `sens`, for the controller defined by the MPC controller object `MPCobj`.

`PerfFunc` must be one of the following strings:

`'ISE'` (integral squared error) for which the performance metric is

$$J = \sum_{i=1}^{\text{Tstop}} \left(\sum_{j=1}^{n_y} (w_j^y e_{yij})^2 + \sum_{j=1}^{n_u} [(w_j^u e_{uij})^2 + (w_j^{\Delta u} \Delta u_{ij})^2] \right)$$

`'IAE'` (integral absolute error) for which the performance metric is

$$J = \sum_{i=1}^{\text{Tstop}} \left(\sum_{j=1}^{n_y} |w_j^y e_{yij}| + \sum_{j=1}^{n_u} (|w_j^u e_{uij}| + |w_j^{\Delta u} \Delta u_{ij}|) \right)$$

`'ITSE'` (integral of time-weighted squared error) for which the performance metric is

$$J = \sum_{i=1}^{\text{Tstop}} i\Delta t \left(\sum_{j=1}^{n_y} (w_j^y e_{yij})^2 + \sum_{j=1}^{n_u} [(w_j^u e_{uij})^2 + (w_j^{\Delta u} \Delta u_{ij})^2] \right)$$

`'ITAE'` (integral of time-weighted absolute error) for which the performance metric is

$$J = \sum_{i=1}^{\text{Tstop}} i\Delta t \left(\sum_{j=1}^{n_y} \left| w_j^y e_{yij} \right| + \sum_{j=1}^{n_u} \left(\left| w_j^u e_{uij} \right| + \left| w_j^{\Delta u} \Delta u_{ij} \right| \right) \right)$$

In the above expressions n_y is the number of controlled outputs and n_u is the number of manipulated variables. e_{yij} is the difference between output j and its setpoint (or reference) value at time interval i. e_{uij} is the difference between manipulated variable `j` and its target at time interval `i`.

The w parameters are non-negative performance weights defined by the structure PerfWeights, which contains the following fields:

`'OutputVariables'`: 1 by n_y vector containing the w_j^y values

`'ManipulatedVariables'`: 1 by n_u vector containing the w_j^u values

`'ManipulatedVariablesRate'`: 1 by n_u vector containing the $w_j^{\Delta u}$ values

If PerfWeights is unspecified, it defaults to the corresponding weights in `MPCobj`. In general, however, the performance weights and those used in the controller have different purposes and should be defined accordingly.

Inputs `Tstop`, `r`, `v`, and `simopt` define the simulation scenario used to evaluate performance. See `sim` for details.

`Tstop` is the integer number of controller sampling intervals to be simulated. The final time for the simulations will be $Tstop \times \Delta t$, where Δt is the controller sampling interval specified in `MPCobj`.

The optional input `utarget` is a vector of n_u manipulated variable targets. Their defaults are zero. Δu_{ij} is the change in manipulated variable `j` and its target at time interval `i`.

The structure variable sens contains the computed sensitivities (partial derivatives of J with respect to the MPCobj tuning weights.) Its fields are

'OutputVariables'	(1 by n_y) sensitivities with respect to MPCobj.Weights.OutputVariables
'ManipulatedVariables'	(1 by n_u) sensitivities with respect to MPCobj.Weights.ManipulatedVariables
'ManipulatedVariablesRate'	(1 by n_u) sensitivities with respect to MPCobj.Weights.ManipulatedVariablesRate

See “Weights” on page 4-6 for details on the tuning weights contained in MPCobj.

[J, sens] = sensitivity(MPCobj,'perf_fun',param1,param2,...) employs a user-defined performance function ’perf_fun’ to define J. Its function definition must be in the form

```
function J = perf_fun(MPCobj, param1, param2, ...)
```

i.e., it must compute J for the given controller and optional parameters param1, param2, ... and it must be on the MATLAB path.

Example

Suppose variable MPCobj contains a default controller definition for a plant with two controlled outputs, three manipulated variables, and no measured disturbances. Compute its performance and sensitivities as follows:

```
PerfFunc = 'IAE';
PerfWts.OutputVariables = [1 0.5];
PerfWts.ManipulatedVariables = zeros(1,3);
PerfWts.ManipulatedVariablesRate = zeros(1,3);
Tstop = 20;
r = [1 0];
v = [];
simopt = mpcsimopt;
utarget = zeros(1,3);
[J, sens] = sensitivity(MPCobj, PerfFunc, PerfWts, Tstop, ...
```

```
    r, v, simopt, utarget)
```

The simulation scenario in the above example uses a constant $r = 1$ for output 1 and $r = 0$ for output 2. In other words, the scenario is a unit step in the output 1 setpoint.

See Also

`mpc`, `sim`

Purpose

Set or modify MPC object properties

Syntax

```
set(MPCobj,'Property',Value)
set(MPCobj,'Property1',Value1,'Property2',Value2,...)
set(MPCobj,'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 4-2 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 4-2 for an overview of legitimate MPC property values.

`set(sys)` displays all assignable properties of sys and their admissible values.

See Also

`mpc`, `get`

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" in the Model Predictive Control Toolbox User's Guide.)

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

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

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 Model Predictive Control Toolbox software is described in "State Estimation" in the Model Predictive Control Toolbox User's Guide. 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` in Control System Toolbox™ software.

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`).

Examples

To design an estimator by pole placement, you can use the commands assuming that the linear system AM=L is solvable.

```
[M,A,Cm]=getestim(MPCobj);
L=place(A',Cm',observer_poles)';
M=A\L;
setestim(MPCobj,M);
```

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 Model Predictive Control Toolbox software 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 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 Model Predictive Control Toolbox software.

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);
```

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`

setname

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`

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 rules:

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 Control System Toolbox software 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 $u(k\text{-}1)$.

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:

$$x(k+1) = Ax(k) + By_m(k) + B_r[r(k);r(k+1);\ldots;r(k+p-1)] + \ldots$$

$$u(k) = Cx(k) + Dy_m(k) + D_r[r(k);r(k+1);\ldots;r(k+p-1)] + \ldots$$

Similarly if the flag `md_preview='on'`, then matrices B_r and D_r multiply the whole measured disturbance sequence:

$$x(k+1) = Ax(k) + \ldots + B_v[v(k);v(k+1);\ldots;v(k+p)] + \ldots$$

$$u(k) = Cx(k) + \ldots + D_v[v(k);v(k+1);\ldots;v(k+p)] + \ldots$$

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 Control System Toolbox software for sensitivity and other linear analysis.

See Also

`ss`, `zpk`

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 Control System Toolbox software for sensitivity and other linear analysis.

See Also

`ss`, `tf`

3

Block Reference

MPC Controller

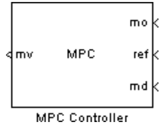
Purpose

Compute MPC control law

Library

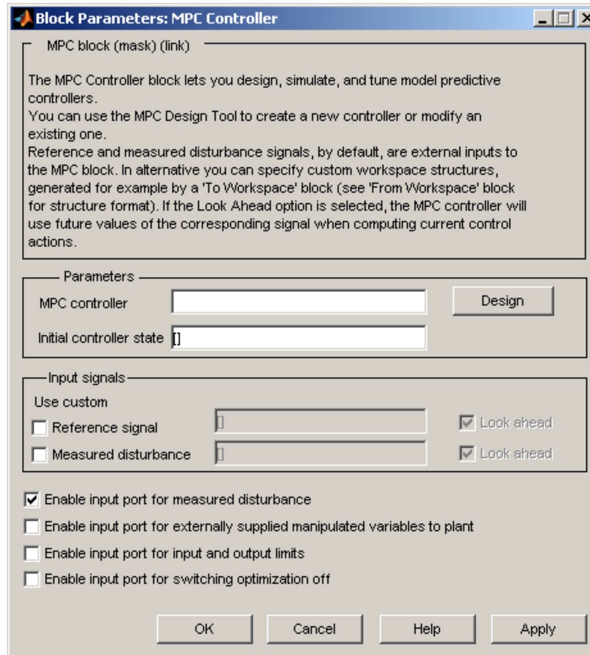
MPC Simulink Library

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.

Dialog Box



MPC controller

You must supply an MPC object that defines your controller. There are two ways to do this. One is to enter the name of an

existing MPC object in the **MPC Controller** edit box. (The object must be in your base workspace.)

The other is to leave the **MPC controller** edit box empty and, with the controller block connected to the plant, click the **Design** button. This constructs a default MPC controller by obtaining a linearized model from the Simulink® diagram. It also opens the design tool so you can modify the default settings.

If you are designing a controller in the design tool, you can see how well it works by running a closed-loop Simulink simulation without exiting the tool. This makes it easier to tune controller parameters.

Initial controller state

Initial state of the MPC controller. This must be a valid `mpcstate` object. If none is supplied, the block uses the default steady-state initial condition.

Reference signal

If you select the check box, the edit box to the right must contain the name of a variable in your workspace that defines the reference signal. This also enables the **Look Ahead** check box. Selecting the **Look Ahead** check box anticipates reference variations and usually improves reference tracking (see "Look Ahead and Signals from the Workspace" in the Model Predictive Control Toolbox User's Guide). If you do not select the **Reference signal** check box, the signal connected to the block `ref` inport supplies the reference values.

Measured disturbance

Provides options for the measured disturbances (for feedforward compensation) in the same way as for the reference signals, above. If you don't supply the measured disturbance here, the signal connected to the block's `md` inport supplies the measured disturbance values.

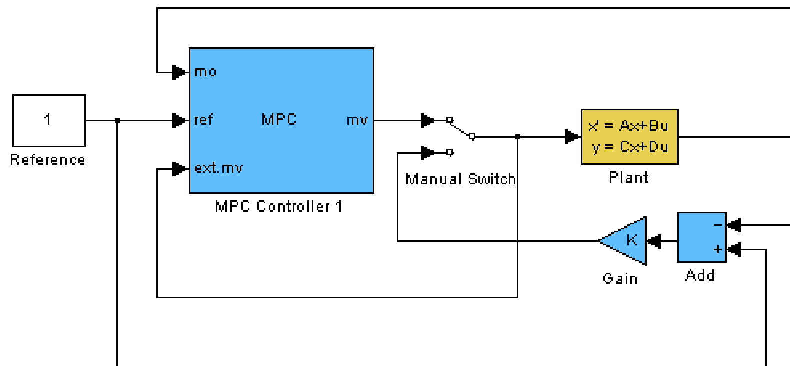
Enable input port for measured disturbance

This option adds an inport (labeled `md`) to which you can connect measured disturbances and for which the controller will provide feedforward compensation.

Enable input port for externally supplied manipulated variables to plant

This check box lets you switch between MPC control and another type of control (e.g., manual control) during a simulation. It adds an inport (labeled `ext.mv`) to which you can connect the actual manipulated variables the plant is receiving. The block uses these in its internal state estimates. The following example shows possible connections. See also the `mpcbumpless` demo.

If the inport is disabled, or it is enabled with no connected signal, the MPC controller assumes that its output is adjusting the plant input. If this is incorrect, the controller's internal state estimate will become inaccurate.

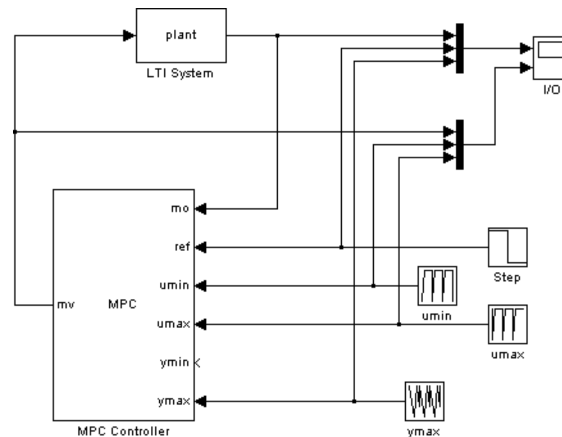


Bumpless Switching Between MPC and Another Controller

Enable input port for input and output limits

This check box adds inports to which you can connect time-varying constraint specifications. Otherwise, the block uses the constant constraint values stored within its MPC object. Example connections appear below. See also the `mpcvarbounds` demo.

When you enable this option, the block interprets an unconnected limit inport, such as `ymin` in the example below, as an unconstrained variable. Also, to prevent numerical difficulties the block enforces a minimum separation of `1e-5` between lower and upper bounds. Further, if a signal connected to a lower-bound port exceeds that connected to the corresponding upper-bound port, the block automatically uses the smaller signal as the lower bound and vice versa.



Enable input port for externally supplied manipulated variables to plant

This check box adds an inport labeled `QP Switch`. If this input signal is zero, the controller behaves normally. If this input becomes nonzero, it turns off the controller's optimization calculations and sets the controller output to zero. This saves computational effort when the controller output is not needed, e.g., the system has been placed in manual operation or another controller has taken over. The controller continues to update its internal state estimate in the usual way, however, so it is ready to resume optimization calculations whenever the `QP Switch` signal returns to zero.

Note The MPC Controller block is a discrete-time block with sampling time inherited from 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`

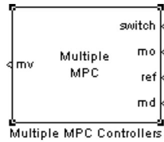
Purpose

Simulate switching between multiple MPC controllers

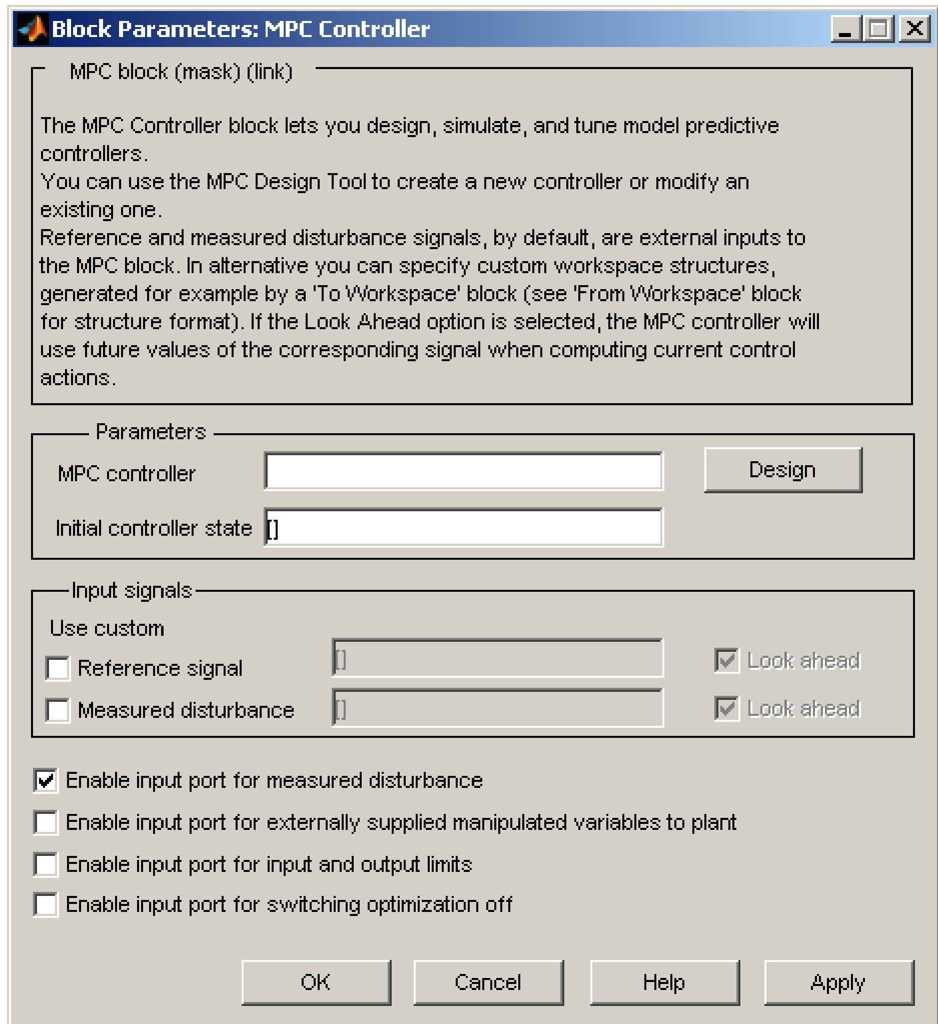
Library

MPC Simulink Library

Description



The Multiple MPC Controllers block receives the current measured output, reference signal, and measured disturbance signal, and solves a quadratic program to calculate the optimal manipulated variables. It also receives a switching signal that designates which of two or more controllers is to perform the calculation. The block contains these controllers as MPC objects, each of which is designed for a particular operating region of a nonlinear plant.



Dialog Box

MPC Object List

The table is an ordered list of MPC objects. The first row designates the controller to be used when the `switch` input equals one, the second designates the controller to be used when the

`switch` input equals two, and so on. These must refer to objects that already exist in your base workspace.

Note After entering each MPC object name, type **Enter**. Also type **Enter** after editing an object name.

Use the **Add** and **Delete** buttons to add and remove rows. When deleting, indicate the row(s) to delete using the **Delete It** checkbox.

When the edit box is empty and the block is connected to the plant, clicking the **Design** button constructs a default MPC controller by obtaining a linearized plant model from the Simulink diagram. It also opens the design tool so you can modify the default behavior.

You can also start the design tool by selecting one or more MPC objects using the **Design It** checkbox, and then clicking the **Design** button. All selected MPC objects will be loaded into the design tool where you can review and edit their properties.

Initial controller state

Initial state of each MPC object in the ordered list. Each must be a valid `mpcstate` object. If none is supplied, the default is a steady-state initial condition.

Reference signal

If you select the check box, the edit box to the right must contain the name of a variable in your workspace that defines the reference signal. This also enables the **Look Ahead** check box. Selecting the **Look Ahead** check box anticipates reference variations and usually improves reference tracking (see "Look Ahead and Signals from the Workspace" in the Model Predictive Control Toolbox User's Guide). If you do not select the **Reference signal** check box, the signal connected to the block `ref` inport supplies the reference values.

Measured disturbance

Provides options for the measured disturbances (for feedforward compensation) in the same way as for the reference signals, above. If you don't supply the measured disturbance here, the signal connected to the block's `md` inport supplies the measured disturbance values.

Enable input port for measured disturbance

This option adds an inport (labeled `md`) to which you can connect measured disturbances and for which the controller will provide feedforward compensation.

Enable input port for externally supplied manipulated variables to plant

This check box lets you switch between MPC control and another type of control (e.g., manual control) during a simulation. It adds an inport (labeled `ext.mv`) to which you can connect the actual manipulated variables the plant is receiving. See MPC Controller for more details.

Enable input port for input and output limits

This check box adds inports to which you can connect time-varying constraint specifications. Otherwise, the block uses the constant constraint values stored within its MPC object.

When you enable this option, the block interprets an unconnected limit inport, such as `ymin` in the example below, as an unconstrained variable. Also, to prevent numerical difficulties the block enforces a minimum separation of `1e-5` between lower and upper bounds. Further, if a signal connected to a lower-bound port exceeds that connected to the corresponding upper-bound port, the block automatically uses the smaller signal as the lower bound and vice versa.

See MPC Controller for more details.

See Also

`mpc`, `mpcmove`, `mpcstate`

4

Object Reference

- "MPC Controller Object" on page 4-2
- "MPC Simulation Options Object" on page 4-14
- "MPC State Object" on page 4-16

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 the following table (MPC Controller Object on page 4-2).

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
`MPCData` (private)	Matrices for the QP problem and other accessorial data
`Version` (private)	Model Predictive Control 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 the following table (Structure ManipulatedVariables on page 4-3), where p denotes the prediction horizon.

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`
`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`

Structure ManipulatedVariables (Continued)

Field Name	Content	Default
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 the following table (Structure OutputVariables on page 4-4), where p denotes the prediction horizon.

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

Structure OutputVariables (Continued)

Field Name	Content	Default
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 Model Predictive Control Toolbox software is a collection of integrators driven by white noise on measured outputs (see "Output Disturbance Model"in the Model Predictive Control Toolbox User's Guide). 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 Model Predictive Control Toolbox software, 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`).

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 the following table (Structure DisturbanceVariables on page 4-6).

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 QP weighting matrices. Unlike the `InputSpecs` and `OutputSpecs`, which are arrays of structures, `weights` is a

single structure containing four fields. The values of these fields depend on whether you are using the standard quadratic cost function (Equation (2–3)) or the alternative cost function (Equation (2–5)).

Standard Cost Function

The table below, Weights for the Standard Cost Function (MATLAB® Structure) on page 4-7, lists the content of the four fields where p denotes the prediction horizon, n_u the number of manipulated variables, n_y the number of output variables.

The fields `ManipulatedVariables`, `ManipulatedVariablesRate`, and `OutputVariables` are arrays with n_u, n_u, and n_y columns, respectively. If weights are time invariant, then `ManipulatedVariables`, `ManipulatedVariablesRate`, and `OutputVariables` are row vectors. However, for time-varying weights, each field is a matrix with up to p rows. If the number of rows is less than the prediction horizon, p, the object constructor duplicates the last row to create a matrix with p rows.

Weights for the Standard Cost Function (MATLAB Structure)

Field Name	Content	Default
`ManipulatedVariables` (or `MV` or `Manipulated` or `Input`)	(1 to p)-by-n_u dimensional array of input weights	`zeros(1,nu)`
`ManipulatedVariablesRate` (or `MVRate` or `ManipulatedRate` or `InputRate`)	(1 to p)-by-n_u dimensional array of input-rate weights	`0.1*ones(1,nu)`
`OutputVariables` (or `OV` or `Controlled` or `Output`)	(1 to p)-by-n_y dimensional array of output weights	`1` (The default for output weights is the following: if $n_u \geq n_y$, all outputs are weighted with unit weight; if $n_u < n_y$, n_u outputs are weighted with unit weight (with preference given to measured outputs), while the remaining outputs receive zero weight.)

Weights for the Standard Cost Function (MATLAB Structure) (Continued)

Field Name	Content	Default
ECR	Weight on the slack variable ε used for softening the constraints	1e5*(max weight)

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" in the Model Predictive Control Toolbox User's Guide).

Alternative Cost Function

You can specify off-diagonal Q and R weight matrices in the cost function. To accomplish this, you must define the fields `ManipulatedVariables`, `ManipulatedVariablesRate`, and `OutputVariables` as cell arrays, each containing a single positive-semi-definite matrix of the appropriate size. Specifically, `OutputVariables` must be a cell array containing the n_y-by-n_y Q matrix, `ManipulatedVariables` must be a cell array containing the n_u-by-n_u R_u matrix, and `ManipulatedVariablesRate` must be a cell array containing the n_u-by-n_u $R_{\Delta u}$ matrix (see Equation (2–5)) and the demo `mpcweightsdemo`). You can abbreviate the field names as shown in Weights for the Standard Cost Function (MATLAB® Structure) on page 4-7. You can also use diagonal weights (as defined in Weights for the Standard Cost Function (MATLAB® Structure) on page 4-7) for one or more of these fields. If you omit a field, the object constructor uses the defaults shown in Weights for the Standard Cost Function (MATLAB® Structure) on page 4-7.

For example, you can specify off-diagonal weights, as follows

```
MPCobj.Weights.OutputVariables={Q};
MPCobj.ManipulatedVariables={Ru};
MPCobj.ManipulatedVariablesRate={Rdu};
```

where `Q`=Q. `Ru`=R_u, and `Rdu`=$R_{\Delta u}$ are positive semidefinite matrices.

Note You cannot specify off-diagonal time-varying weights.

Model

The property `Model` specifies plant, input disturbance, and output noise models, and nominal conditions, according to the model setup described in "Model Used for State Estimation" in the Model Predictive Control Toolbox User's Guide. It is specified through a structure containing the fields reported in Structure Model Describing the Models Used by MPC on page 4-9.

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
`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 Nominal Values at Operating Point on page 4-11.

Note Direct feedthrough from manipulated variables to any output in `Model.Plant` is not allowed. See "Prediction Model" in the Model Predictive Control Toolbox User's Guide.

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 Input Groups in Plant Model on page 4-10 and Output Groups in Plant Model on page 4-10.

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

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 Nominal Values at Operating Point on page 4-11 (see "Offsets" in the Model Predictive Control Toolbox User's Guide).

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 the following table (Optimizer Properties on page 4-12).

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" in the Model Predictive Control Toolbox User's Guide).

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` and `setmpcdata`.

Version

`Version` is a private property indicating the Model Predictive Control 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 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 the following table (MPC Simulation Options Properties on page 4-14).

MPC Simulation Options Properties

Property	Description
`PlantInitialState`	Initial state vector of the 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 the wait bar (`'on'` or `'off'`).
`MVSignal`	Sequence of manipulated variables (with offsets) for open-loop simulation (no MPC action).
`OpenLoop`	Perform 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.

MPC State Object

The `mpcstate` object type contains the state of an MPC controller. Its properties are listed in MPC State Object Properties on page 4-16.

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 $u(k\text{-}1)$. 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`.

Index

C

constraints
    specification 4-3

D

DC gain 2-2
disturbances
    input model 2-10
    internal state 2-28
    output model 2-13
    type definition 2-45

E

estimation 2-40
    gain 2-40
    model extraction 2-7
    See also observer

I

inputs
    disturbance model 2-10

M

memory 2-4
models
    input disturbance 2-10
    mismatch 2-26
    mismatch in simulations 4-15
    output disturbance retrieval 2-13
MPC Controller, Simulink
    initial state 3-3
MPC state 2-28
mpctool 2-29

N

nominal conditions
    structure 4-11

O

observer 4-13
    gain 2-40
    initialization 4-13
    model 2-7
    See also estimation
offset 4-11
optimal trajectory 2-20
outputs
    disturbance model retrieval 2-13

P

plants
    initial state 4-14
    mismatch 2-26
    mismatch in simulations 4-15

S

sensitivity 2-35
Simulink
    block 3-2
    initial state 3-9
state
    controller, definition 2-28
    observer 2-7
        See also estimation
states
    initial 4-14
steady-state 2-2

U

unconstrained control 2-2
unmeasured disturbances
    input model 2-10
    output model 2-13

W
weights
specification 4-6