# FORCESPR MATHWORKS MPC TOOLBOX PLUGIN

#### Uros Markovic Optimization Specialist

2022 American Control Conference Atlanta, GA, USA / June 2022



# MATHWORKS MPC TOOLBOX PLUGIN

- Use of **FORCES**PRO solver in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Enables code deployment of the **FORCES**PRO solver on real-time hardware from within MATLAB and Simulink
- Leverages the **powerful design capabilities** of MPC Toolbox and the **computational performance** of **FORCES**PRO
- Additional features such as Simulink blocks that can generate code runnable on embedded targets such as dSpace
- Generated code is highly optimized for fast computations and low memory footprint
- Both linear and nonlinear MPC Toolbox plugins available





#### MATHWORKS MPC TOOLBOX PLUGIN

- Use of **FORCES**PRO solver in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Enables code deployment of the **FORCES**PRO solver on real-time hardware from within MATLAB and Simulink
- Leverages the **powerful design capabilities** of MPC Toolbox and the **computational performance** of **FORCES**PRO
- Additional features such as Simulink blocks that can generate code runnable on embedded targets such as dSpace
- Generated code is highly optimized for fast computations and low memory footprint
- Both linear and nonlinear MPC Toolbox plugins available



embotec



# LINEAR MPC TOOLBOX PLUGIN



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* for using the **FORCES**PRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - **mpcToForces** for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - **mpcmoveForces** for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options

% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - *mpcToForcesOptions* an auxiliary file exposed to the users for generating different solver options

LTI MPC output of mpc(...)

% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - *mpcToForcesOptions* an auxiliary file exposed to the users for generating different solver options

output of mpcToForcesOptions(...)

% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - *mpcToForcesOptions* an auxiliary file exposed to the users for generating different solver options

output of mpcToForcesOptions(...)

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);
```

```
% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

Two types of **QP solvers** can be generated via *mpcToForces* (provided via *mpcToForcesOptions* arguments):

- **"sparse"** solver corresponds to a multi-stage formulation as in low-level interface (customForcesSparseQP.m)
- **"dense"** solver corresponds to a single-stage QP with inequality constraints (*customForcesDenseQP.m*) **embotech**

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - mpcToForces for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);
% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```





- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);
```

```
% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

optimal manipulated variables at current solve time instant



- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* for calling the generated solver on a specific MPC problem instance
  - mpcCustomSolver for using the FORCESPRO dense QP solver as a custom solver
  - mpcToForcesOptions an auxiliary file exposed to the users for generating different solver options



#### SIMULINK INTERFACE

• Both **sparse** and **dense** solvers **can be used inside Simulink** 



#### DENSE QP SIMULINK INTERFACE

• Both **sparse** and **dense** solvers **can be used inside Simulink** 

• Dense QP formulation usable from the shipped Simulink MPC controller block directly

```
% Generate a custom dense FORCESPRO solver
options = mpcToForcesOptions('dense');
mpcToForces(mpcobj, options);
```

% Enable the settings in the MPC object mpcobj.Optimizer.CustomSolver = true; mpcobj.Optimizer.CustomSolverCodeGen = true;



- Both sparse and dense solvers can be • used inside Simulink
- **Sparse QP** solver also available via the • MPC Toolbox and FORCESPRO MPC **block** in the Library browser
- Configuration done via Simulink UI •
- All MATLAB plugin features are • available through this Simulink block

Simulink Library Browser	-		×	Block Parameters: FORCES MPC (Sparse QP)	2
				ForcesMPC_SparseQP (mask) (link)	
				Simulate MPC controller and generate code using FORCES QP so	lver.
FORCES PRO MPC Blocks          FORCES PRO MPC Blocks <ul> <li>Additional Math &amp; Discrete</li> <li>Commonly Used Blocks</li> <li>Continuous</li> <li>Dashboard</li> <li>Discrete</li> <li>Logic and Bit Operations</li> <li>Lookup Tables</li> <li>Math Operations</li> <li>Model Verification</li> <li>Model Verification</li> <li>Model Verification</li> <li>Model Verification</li> <li>Model Verification</li> <li>Signal Attributes</li> <li>Signal Routing</li> <li>Sinks</li> <li>Sources</li> <li>String</li> <li>User-Defined Functions</li> <li>Quick Insert</li> <li>Control System Toolbox</li> <li>Embedded Coder</li> <li>FORCES PRO MPC Blocks</li> <li>HDL Coder</li> <li>Model Predictive Control Toolbox</li> <li>Simulink XD Animation</li> <li>Simulink Real-Time</li> <li>Stateflow</li> <li>Recently Used</li> </ul>	FORCES PRO Nonlinear MPC _mv CCES Nonlinear	mv MPC		Simulate MPC controller and generate code using FORCES QP so         In MATLAB, use the following commands to generate custom FOI solver from your MPC controller (mpcobj):         options = mpcToForcesOptions('sparse');         [coredata, statedata, online features via the "options".         Afterwards, specify "coredata" and "statedata" in the block dialog additional input and output signals in the block that must be cons "options".         Parameters         Core Data       coredata         Core Data       coredata         Additional Signal Configuration         Ø Measured disturbance (md)         External manipulated variable (ext.mv)         References for manipulated variables (mv.target)         State Estimation         Use custom state estimation (x[k]k])         Online Constraints         Lower MV Limits (umin)       Upper MV Limits (uma         Lower MV Limits (umin)       Upper MV Limits (uma         Online Weights       MV Weights (u.wt)         Ø MVRate Limits (du.wt)       Slack Variable Weight         Additional Output Signals       Optimal control seque         Optimal cost (cost)       Optimal control seque	ver. tCES QP ons); and enable istent with 



 $\times$ 

- Both **sparse** and **dense** solvers **can be used inside Simulink**
- Sparse QP solver also available via the MPC Toolbox and FORCESPRO MPC block in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)

📲 Simulink Library Browser	- 🗆 🗙 🔤 Block Parameters: FORCES MPC (Sparse QP)
	-ForcesMPC_SparseQP (mask) (link)
🖓 Scope 🗸 🖓 🔻 🔄 🔻 🕂 🖓	Simulate MPC controller and generate code using FORCES QP solver.
<ul> <li>FORCES PRO MPC Blocks</li> <li>✓ Simulink</li> <li>&gt; Additional Math &amp; Discrete Commonly Used Blocks Continuous Dashboard Discrete Logic and Bit Operations Lookup Tables Math Operations Messages &amp; Events Model Verification Model-Wide Utilities Ports &amp; Subsystems Signal Attributes Signal Routing Sinks Sources String User-Defined Functions</li> <li>&gt; Quick Insert</li> <li>&gt; Control System Toolbox</li> <li>&gt; Embedded Coder FORCES PRO MPC Blocks</li> <li>&gt; HDL Coder</li> <li>&gt; Model Predictive Control Toolbox</li> <li>&gt; Simulink Coder</li> <li>&gt; Simulink Real-Time Stateflow Recently Used</li> </ul>	In MATLAB, use the following commands to generate custom FORCES QP solver from your MPC controller (mpcob): options? mpcToForceS(mpcob); options?);         ref       FORCES PRO Nonlinear MPC         FORCES Nonlinear MPC       Afterwards, specify "coredata" and "statedata" in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals in the block dialog and enal additional input and output signals         See QP)       FORCES Nonlinear MPC         Core Data       State Estimation         Initial State Data       State State data         Initial State Data       State Estimation         Use custom state estimation (x[k]k])       Online Constraints         Lower OV Limits (umin)       Upper MV Limits (umax)         Lower MVRate Limits (durinin)       Upper MV Limits (durina)



- Both **sparse** and **dense** solvers **can be used inside Simulink**
- Sparse QP solver also available via the MPC Toolbox and FORCESPRO MPC block in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)
  - external manipulated vars (ext.mv)

📲 Simulink Library Browser		- 🗆	×	Block Parameters: FORCES MPC (Spar	se QP)
				ForcesMPC_SparseQP (mask) (link)	
	* = 0			Simulate MPC controller and generate	code using FORCES QP solver.
FORCES PRO MPC Blocks  FORCES PRO MPC Blocks  Simulink Additional Math & Discrete Commonly Used Blocks Continuous Dashboard Discontinuities Discrete Logic and Bit Operations Lookup Tables Math Operations Messages & Events Model Verification Model-Wide Utilities Ports & Subsystems Signal Attributes Signal Attributes Signal Autriputes Signal Routing Sinkis Sources String User-Defined Functions Quick Insert Control System Toolbox Embedded Coder FORCES PRO MPC Blocks HDL Coder Model Predictive Control Toolbox Simulink 2D Animation Simulink Real-Time Stateflow Recently Used	FORCES MPC (Sparse QP) FORCES MPC (Sparse QP) FORCES MPC (Sparse QP) FORCES NOT	S PRO mv		Simulate MPC controller and generate In MATLAB, use the following commal solver from your MPC controller (mpco options = mpcToForcesOptions('sp [coredata, statedata, onlinedata] = You can enable additional online featu Afterwards, specify "coredata" and "st additional input and output signals in "options". Parameters Core Data coredata Initial State Data statedata Additional Signal Configuration Measured disturbance (md) External manipulated variable (ext. References for manipulated variable) State Estimation Use custom state estimation (x[k]k Online Constraints Lower OV Limits (umin) Lower MV Limits (umin) Online Weights OV Weights (y.wt) MVRate Weights (du.wt) Additional Output Signals Optimal cost (cost) Optimization status (qp.status) Estimated current states (est.state	code using FORCES QP solver. uds to generate custom FORCES QP ubj): mpcToForces(mpcobj, options); res via the "options". atedata" in the block dialog and ena- the block that must be consistent w ww) es (mv.target) ]) Upper OV Limits (ymax) Upper MV Limits (ymax) Upper MV Limits (umax) Upper MVRate Limits (dumax) Slack Variable Weight (ecr.wt; Optimal control sequence (m. Optimal output sequence (y.set)
				OK	Cancel Help Ar



- Both sparse and dense solvers can be • used inside Simulink
- **Sparse QP** solver also available via the • MPC Toolbox and FORCESPRO MPC **block** in the Library browser
- Configuration done via Simulink UI .
- All MATLAB plugin features are • available through this Simulink block
  - measured disturbances (*md*) ٠
  - external manipulated vars (ext.mv) ٠
  - ref. for manipulated vars (mv.target) ٠



Apply

 $\times$ 

- Both **sparse** and **dense** solvers **can be** • used inside Simulink
- **Sparse QP** solver also available via the • MPC Toolbox and FORCESPRO MPC **block** in the Library browser
- Configuration done via Simulink UI •
- All MATLAB plugin features are ٠ **available** through this Simulink block
  - measured disturbances (md) ٠
  - external manipulated vars (ext.mv) ٠
  - ref. for manipulated vars (mv.target) ٠
  - custom state estimation •





Apply

Х

- Both sparse and dense solvers can be • used inside Simulink
- **Sparse QP** solver also available via the • MPC Toolbox and FORCESPRO MPC **block** in the Library browser
- Configuration done via Simulink UI •
- All MATLAB plugin features are • available through this Simulink block
  - measured disturbances (*md*) ٠
  - external manipulated vars (ext.mv) ٠
  - ref. for manipulated vars (mv.target) ٠
  - custom state estimation •
  - online weights and constraints ٠

Simulink Library Browser	_		×	🚡 Block Parameters: FORCES MPC (Spa	irse QP)
				ForcesMPC_SparseQP (mask) (link)	
				Simulate MPC controller and generate	e code using FORCES QP solver.
Scope          Scope       Image: Control of the second	x ref FORCES PRO Nonlinear MPC ast_mv FORCES Nonlinear N	mv 4PC		ForcesMPC_SparseQP (mask) (link)         Simulate MPC controller and generate         In MATLAB, use the following commasoliver from your MPC controller (mpy options) = mpCToForcesOptions('s [coredata, statedata, onlinedata]:         You can enable additional online feat         Afterwards, specify "coredata" and "s additional input and output signals in "options".         Parameters         Core Data       coredata         Initial State Data       statedata         Additional Signal Configuration       Measured disturbance (md)         External manipulated variable (ext       References for manipulated variable (ext         State Estimation       Use custom state estimation (x[k]         Online Constraints       Lower OV Limits (ymin)         Lower MV Rate Limits (dumin)       Online Weights         OV Weights (y.wt)       MVRate Weights (du.wt)         Additional Output Signals       Optimal cost (cost)	e code using FORCES QP solver. ands to generate custom FORCES QP cobj): = mpcToForces(mpcobj, options); urres via the "options". statedata" in the block dialog and enable the block that must be consistent with 



Generate FORCESPRO sparse QP solver

% Generate FORCESPRO sparse QP solver options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks options.UseOnlineWeightOV = true; options.UseOnlineWeightMVRate = true; options.UseOnlineWeightECR = true; [coredata, statedata, onlinedata] = mpcToForces(mpcobj, options); Block Parameters: FORCES MPC (Sparse QP) Х ForcesMPC\_SparseQP (mask) (link) Simulate MPC controller and generate code using FORCES QP solver. In MATLAB, use the following commands to generate custom FORCES QP solver from your MPC controller (mpcobj): options = mpcToForcesOptions('sparse'); [coredata, statedata, onlinedata] = mpcToForces(mpcobj, options); You can enable additional online features via the "options". Afterwards, specify "coredata" and "statedata" in the block dialog and enable additional input and output signals in the block that must be consistent with "options". Parameters Core Data coredat Initial State Data statedata Additional Signal Configuration Measured disturbance (md) External manipulated variable (ext.mv) References for manipulated variables (mv.target) State Estimation Use custom state estimation (x[k|k]) Online Constraints Lower OV Limits (ymin) Upper OV Limits (ymax) Lower MV Limits (umin) Upper MV Limits (umax) Lower MVRate Limits (dumin) Upper MVRate Limits (dumax) Online Weights ✓ OV Weights (y.wt) MV Weights (u.wt) MVRate Weights (du.wt) Slack Variable Weight (ecr.wt) Additional Output Signals Optimal cost (cost) Optimal control sequence (my.seq) Optimization status (qp.status) Optimal state sequence (x.seq) Estimated current states (est.state) Optimal output sequence (y.seq) OK Help Apply Cancel



```
• Generate FORCESPRO sparse QP solver
```

• Provide coredata, statedata and onlinedata to the Simulink block

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

Block Parameters: FORCES MPC (Sparse	(QP) X				
ForcesMPC_SparseQP (mask) (link)					
Simulate MPC controller and generate of	ode using FORCES QP solver.				
In MATLAB, use the following commands to generate custom FORCES QP solver from your MPC controller (mpcobj): options = mpcToForcesOptions('sparse'); [coredata, statedata, onlinedata] = mpcToForces(mpcobj, options); You can enable additional online features via the "options".					
Atterwards, specity "coredata" and "sta additional input and output signals in th "options".	tedata" in the block dialog and enable he block that must be consistent with				
Parameters					
Core Data coredata	:				
Initial State Data statedata	:				
Additional Signal Configuration					
Measured disturbance (md)					
External manipulated variable (ext.n	יי)				
References for manipulated variables (mv.target)					
State Estimation					
Use custom state estimation (x[k k])	)				
Online Constraints					
Lower OV Limits (ymin)	Upper OV Limits (ymax)				
Lower MV Limits (umin)	Upper MV Limits (umax)				
Lower MVRate Limits (dumin)	Upper MVRate Limits (dumax)				
Online Weights					
OV Weights (y.wt)	MV Weights (u.wt)				
MVRate Weights (du.wt)	☑ Slack Variable Weight (ecr.wt)				
Additional Output Signals					
Optimal cost (cost)	Optimal control sequence (mv.seq)				
Optimization status (qp.status)	Optimal state sequence (x.seq)				
Estimated current states (est.state)	Optimal output sequence (y.seq)				
ОК	Cancel Help Apply				



```
    Generate FORCESPRO sparse QP solver
```

- Provide coredata, statedata and onlinedata to the Simulink block
- Select md checkbox if MD channels exist in the MPC object

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

Block Parameters: FORCES MPC (Sparse	e QP) X				
ForcesMPC_SparseQP (mask) (link)					
Simulate MPC controller and generate of	code using FORCES QP solver.				
In MATLAB, use the following commands to generate custom FORCES QP solver from your MPC controller (mpcob)): options = mpcToForcesOptions('sparse'); [coredata, statedata, onlinedata] = mpcToForces(mpcobj, options); You can enable additional online features via the "options". Afterwards, specify "coredata" and "statedata" in the block dialog and enable					
"options".					
Parameters					
Core Data coredata					
Initial State Data statedata					
Additional Signal Configuration					
Measured disturbance (md)					
External manipulated variable (ext.n	nv)				
References for manipulated variables (my.target)					
State Estimation					
Use custom state estimation (x[k k]	)				
Online Constraints					
Lower OV Limits (ymin)	Upper OV Limits (ymax)				
Lower MV Limits (umin)	Upper MV Limits (umax)				
Lower MVRate Limits (dumin)	Upper MVRate Limits (dumax)				
Online Weights					
OV Weights (v.wt)	MV Weights (u.wt)				
MVRate Weights (du.wt)	Slack Variable Weight (ecr.wt)				
Additional Output Signals					
Optimal cost (cost)	Optimal control sequence (mv.seq)				
Optimization status (qp.status)	Optimal state sequence (x.seq)				
Estimated current states (est.state)	Optimal output sequence (y.seq)				
ОК	Cancel Help Apply				



```
    Generate FORCESPRO sparse QP solver
```

- Provide coredata, statedata and onlinedata to the Simulink block
- Select md checkbox if MD channels exist in the MPC object
- Select x[k|k] checkbox for using a custom state estimator

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

Block Parameters	s: FORCES MPC (Sparse	≘QP)		×
ForcesMPC_Sparse	eQP (mask) (link)			
Simulate MPC cont	troller and generate o	ode using FC	RCES QP solv	er.
In MATLAB, use th solver from your M options = mpcT [coredata, state You can enable ad Afterwards, specify additional input an	e following command IPC controller (mpcol "oForcesOptions('spai data, onlinedata] = i ditional online featur y "coredata" and "sta d output signals in th	ds to generate oj): rse'); mpcToForces( es via the "opi tedata" in the ne block that r	e custom FORG mpcobj, option tions". block dialog a nust be consis	CES QP ns); and enable stent with
"options".				
Parameters				
Core Data cored	ata			:
Initial State Data	statedata			:
Additional Signal C Measured distu External manipu References for	Configuration rbance (md) ulated variable (ext.n manipulated variable	nv) s (mv.target)		
State Estimation				
Use custom sta	te estimation (x[k k]	)		
Online Constraints				
Lower OV Limit	s (ymin)	Upper OV	Limits (ymax)	)
Lower MV Limit	rs (umin)	Upper MV	Limits (umax	)
Lower MVRate	Limits (dumin)	Upper MV	Rate Limits (d	lumax)
Online Weights				
✓ OV Weights (y.	wt)	MV Weigh	nts (u.wt)	
MVRate Weight	s (du.wt)	Slack Vari	able Weight (	ecr.wt)
Additional Output	Signals			
Optimal cost (c	ost)	Optimal c	ontrol sequen	ce (mv.seq)
Optimization st	atus (qp.status)	Optimal s	tate sequence	(x.seq)
Estimated curre	ent states (est.state)	Optimal o	utput sequen	ce (y.seq)
	ОК	Cancel	Help	Apply



- Generate FORCESPRO sparse QP solver
- Provide coredata, statedata and onlinedata to the Simulink block
- Select md checkbox if MD channels exist in the MPC object
- Select x[k|k] checkbox for using a custom state estimator
- Monitor **qp.status** output for feasibility of the MPC block solution

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

Block Parameters: FORCES MPC (Spars	e QP) X			
ForcesMPC_SparseQP (mask) (link)				
Simulate MPC controller and generate	code using FORCES QP solver.			
In MATLAB, use the following commands to generate custom FORCES QP solver from your MPC controller (mpcobj): options = mpcToForcesOptions('sparse'); [coredata, statedata, onlinedata] = mpcToForces(mpcobj, options); You can enable additional online features via the "options".				
additional input and output signals in t "options".	tedata" in the block dialog and enable he block that must be consistent with			
Parameters				
Core Data coredata	1			
Initial State Data statedata	:			
Additional Signal Configuration Measured disturbance (md) External manipulated variable (ext.r References for manipulated variable	mv) 25 (mv.target)			
State Estimation				
Use custom state estimation (x[k k]	))			
Online Constraints				
Lower OV Limits (ymin)	Upper OV Limits (ymax)			
Lower MV Limits (umin)	Upper MV Limits (umax)			
Lower MVRate Limits (dumin)	Upper MVRate Limits (dumax)			
Online Weights				
OV Weights (y.wt)	MV Weights (u.wt)			
MVRate Weights (du.wt)	☑ Slack Variable Weight (ecr.wt)			
Additional Output Signals				
Optimal cost (cost)	Optimal control sequence (mv.seq)			
Optimization status (qp.status)	Optimal state sequence (x.seq)			
Estimated current states (est.state)	Optimal output sequence (y.seq)			
ОК	Cancel Help Apply			



#### Generate FORCESPRO sparse QP solver

- Provide coredata, statedata and onlinedata to the Simulink block
- Select md checkbox if MD channels exist in the MPC object
- Select x[k|k] checkbox for using a custom state estimator
- Monitor **qp.status** output for feasibility of the MPC block solution

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

# % Simulation open\_system('forcesmpc\_onlinetuning'); % Open Simulink model sim(mdl); % Start simulation





• Generate FORCESPRO sparse QP solver for the targeted platform



```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```





- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**
- options.ForcesTargetPlatform needs to be specified



```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```





- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**
- options.ForcesTargetPlatform needs to be specified
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block



```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

```
% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```





- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**
- options.ForcesTargetPlatform needs to be specified
- Provide coredata, statedata and onlinedata to the Simulink block

More details on deployment to dSPACE MicroAutobox II can be found here

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');
```

% Specify online weights on outputs, input rates and ECR slacks options.UseOnlineWeightOV = true; options.UseOnlineWeightMVRate = true; options.UseOnlineWeightECR = true; options.ForcesTargetPlatform = 'dSPACE-MABII'; [coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);







NONLINEAR MPC TOOLBOX PLUGIN



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox

Depending on the chosen object, the nonlinear MPC plugin consists of two API methods:

- nImpcToForces / nImpcMultistageToForces generates a FORCESPRO nonlinear solver from a nonlinear MPC (nImpc) or a nonlinear MPC multistage (nImpcMultistage) object designed by the MPC Toolbox
- nImpcmoveForces / nImpcmoveForcesMultistage calls the generated solver to calculate optimal control actions



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox

% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox

nlmpc / nlmpcMultistage object

% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);
% Calling the solver
[mv, onlinedata, info] = nlmpcmoveForces(coredata, x, mv, onlinedata); % MPC Toolbox interface
```



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);
```

#### % Calling the solver

[mv, onlinedata, info] = nlmpcmoveForces(coredata, x, mv, onlinedata); % MPC Toolbox interface
[mv, onlinedata, info] = nlmpcmove\_CstrSolver(x, mv, onlinedata); % MEX function (speed up)



- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox







- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- FORCESPRO automatically generates jacobian functions (via automatic differentiation tool CasADi)
- State and output jacobian functions that may be provided within the MPC prediction model will be ignored
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin can also use the MATLAB Symbolic Math Toolbox



#### EXAMPLE: LANE FOLLOWING

- Lane-following system keeps the vehicle traveling along the centerline of a highway lane, while maintaining a user-set velocity
- Manipulates both the longitudinal acceleration and the front steering angle of the vehicle in order to:
  - keep the lateral deviation e<sub>1</sub> and relative yaw angle e<sub>2</sub>
     small
  - keep the longitudinal velocity V<sub>x</sub> close to a driver-set velocity
  - **balance the above two goals** when they cannot be met simultaneously



Detailed example description and MATLAB code are available for download here





Model contains **four** main components:

 Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ







- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle





Model contains **four** main components:

- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle

 $\dot{e}_1 =$ longitudinal velocity \*  $e_2$  + lateral velocity  $\dot{e}_2 =$  yaw rate - longitudinal velocity \* curvature



- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle

$$\dot{e}_1 = V_x * e_2 + V_y$$
$$\dot{e}_2 = \dot{\psi} - u_3$$



- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle
- Lane-following controller: simulates nonlinear MPC





- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle
- Lane-following controller: simulates nonlinear MPC
  - Comprises seven states, three outputs, and two inputs

```
function y = LaneFollowingOutputFcn(x,u)
    % Output
    y = [x(3); ...
        x(5); ...
        x(6)+x(7)];
end
```

% States x = [	lateral velocity (Vy)
%	yaw rate (psi_dot)
%	longitudinal velocity (Vx)
%	longitudinal acceleration (Vx_dot)
%	lateral deviation (e1)
%	relative yaw angle (e2)
%	<pre>output disturbance of relative yaw angle (xOD)];</pre>
%	
% Inputs u = [	acceleration
%	steering angle
%	road curvature * Vx (measured disturbance)
%	white noise (unmeasured disturbance)];



- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle
- Lane-following controller: simulates nonlinear MPC
  - Comprises seven states, three outputs, and two inputs
  - Model has two MV signals: acceleration and steering

<pre>% States x = [ % % % % % % % % %</pre>	<pre>lateral velocity (Vy) yaw rate (psi_dot) longitudinal velocity (Vx) longitudinal acceleration (Vx_dot) lateral deviation (e1) relative yaw angle (e2) output disturbance of relative yaw angle (xOD)]:</pre>
% % Inputs u = [ % % %	<pre>acceleration steering angle road curvature * Vx (measured disturbance) white noise (unmeasured disturbance)];</pre>



- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle
- Lane-following controller: simulates nonlinear MPC
  - Comprises seven states, three outputs, and two inputs
    - Model has two MV signals: acceleration and steering
    - **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
    - Unmeasured disturbance modeled by white noise

% States x = [	lateral velocity (Vy)
%	yaw rate (psi_dot)
%	longitudinal velocity (Vx)
%	longitudinal acceleration (Vx_dot)
%	lateral deviation (e1)
%	relative yaw angle (e2)
%	output disturbance of relative yaw angle (xOD)];
%	
% Inputs u = [	acceleration
%	steering angle
%	road curvature * Vx (measured disturbance)
%	<pre>white noise (unmeasured disturbance)];</pre>



```
% States x = [ lateral velocity (Vy)
% yaw rate (psi_dot)
% longitudinal velocity (Vx)
% longitudinal acceleration (Vx_dot)
% lateral deviation (e1)
% relative yaw angle (e2)
% output disturbance of relative yaw angle (xOD)];
%
% Inputs u = [ acceleration
% steering angle
% road curvature * Vx (measured disturbance)
% white noise (unmeasured disturbance)];
```

- Lane-following controller: simulates nonlinear MPC
  - Comprises seven states, three outputs, and two inputs
    - Model has two MV signals: acceleration and steering
    - **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
    - Unmeasured disturbance modeled by white noise

```
function dxdt = LaneFollowingStateFcn(x,u)
% Vehicle Parameters
m = 1575; % Mass of car
Iz = 2875; % Moment of inertia about Z axis
If = 1.2; % Distance between Center of Gravity and Front axle
Ir = 1.6; % Distance between Center of Gravity and Rear axle
Cf = 19000; % Cornering stiffness of the front tires (N/rad)
Cr = 33000; % Cornering stiffness of the rear tires (N/rad).
tau = 0.2; % Time constant
```

```
% State Equations
a1 = -(2*Cf+2*Cr)/m/x(3);
a2 = -(2*Cf*lf-2*Cr*lr)/m/x(3) - x(3);
a3 = -(2*Cf*lf-2*Cr*lr)/Iz/x(3);
a4 = -(2*Cf*lf^2+2*Cr*lr^2)/Iz/x(3);
b1 = 2 * Cf/m;
b2 = 2*Cf*lf/Iz;
dxdt = [a1*x(1) + a2*x(2) + b1*u(2); % Vy
        a3*x(1) + a4*x(2) + b2*u(2); % psi_dot
        x(2)*x(1) + x(4);
                                      % Vx
        (1/tau)*(-x(4) + u(1));
                                      % Vx_dot
        x(1) + x(3) * x(6);
                                      % e1
        x(2) - u(3);
                                      % e2
        u(4)];
                                      % xOD
```



- Vehicle dynamics: applies 3<sup>rd</sup>-order bicycle model of lateral vehicle dynamics and approximates longitudinal dynamics using a time constant τ
- Sensor dynamics: approximates a sensor such as a camera to calculate the lateral deviation and relative yaw angle
- Lane-following controller: simulates nonlinear MPC
- **Curvature previewer:** detects the curvature at the current time step and the **curvature sequence** over the prediction horizon of the MPC controller







#### SYSTEM OVERVIEW



Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- Unmeasured disturbance modeled by white noise





Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- Unmeasured disturbance modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);
% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;
% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```





Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- Unmeasured disturbance modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);
% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;
% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
```

```
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```

% Set constraints for	manipulated variables
nlobj.MV(1).Min = -3;	<pre>% Maximum acceleration 3 m/s^2</pre>
<pre>nlobj.MV(1).Max = 3;</pre>	% Minimum acceleration -3 m/s^2
<pre>nlobj.MV(2).Min = -1.1</pre>	.3; % Minimum steering angle -65 deg
<pre>nlobj.MV(2).Max = 1.13</pre>	; % Maximum steering angle 65 deg



#### Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- Unmeasured disturbance modeled by white noise

#### % Set constraints for manipulated variables

```
nlobj.MV(1).Min = -3; % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3; % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13; % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13; % Maximum steering angle 65 deg
```

#### % Set scale factors

```
nlobj.OV(1).ScaleFactor = 15; % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5; % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5; % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6; % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2; % Range of curvature
```

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);
% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;
% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
```

nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';

embote

#### Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);
```

```
% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;
```

% Specify state and output functions for the plant model nlobj.Model.StateFcn = 'LaneFollowingStateFcn'; nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';

#### % Set constraints for manipulated variables

```
nlobj.MV(1).Min = -3; % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3; % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13; % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13; % Maximum steering angle 65 deg
```

#### % Set scale factors

```
nlobj.OV(1).ScaleFactor = 15; % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5; % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5; % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6; % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2; % Range of curvature
```

% Specify the weights in the standard MPC cost function nlobj.Weights.OutputVariables = [1 1 0];



#### Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);
```

```
% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;
```

% Specify state and output functions for the plant model nlobj.Model.StateFcn = 'LaneFollowingStateFcn'; nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';

#### % Set constraints for manipulated variables

```
nlobj.MV(1).Min = -3; % Maximum acceleration 3 m/s<sup>2</sup>
nlobj.MV(1).Max = 3; % Minimum acceleration -3 m/s<sup>2</sup>
nlobj.MV(2).Min = -1.13; % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13; % Maximum steering angle 65 deg
```

#### % Set scale factors

```
nlobj.OV(1).ScaleFactor = 15; % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5; % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5; % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6; % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2; % Range of curvature
```

% Specify the weights in the standard MPC cost function nlobj.Weights.OutputVariables = [1 1 0];

% Penalize acceleration change more for smooth driving experience nlobj.Weights.ManipulatedVariablesRate = [0.3 0.1];



#### Lane-following controller specification:

- Comprises seven states, three outputs, and two inputs
- Model has two MV signals: acceleration and steering
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- Unmeasured disturbance modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);
```

```
% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;
```

% Specify state and output functions for the plant model nlobj.Model.StateFcn = 'LaneFollowingStateFcn'; nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';

#### % Set constraints for manipulated variables

```
nlobj.MV(1).Min = -3; % Maximum acceleration 3 m/s<sup>2</sup>
nlobj.MV(1).Max = 3; % Minimum acceleration -3 m/s<sup>2</sup>
nlobj.MV(2).Min = -1.13; % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13; % Maximum steering angle 65 deg
```

#### % Set scale factors

```
nlobj.OV(1).ScaleFactor = 15; % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5; % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5; % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6; % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2; % Range of curvature
```

% Specify the weights in the standard MPC cost function nlobj.Weights.OutputVariables = [1 1 0];

% Penalize acceleration change more for smooth driving experience nlobj.Weights.ManipulatedVariablesRate = [0.3 0.1];

```
% Generate FORCESPRO nonlinear solver
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
[coredata, onlinedata] = nlmpcToForces(nlobj,options);
```



# SIMULINK INTEGRATION

- Generated FORCESPRO NLMPC Simulink block can now be used seamlessly with the model
- Available in the Model Predictive Control Toolbox
   section of the Simulink Library Browser





# SIMULINK INTEGRATION

- Generated FORCESPRO NLMPC Simulink block can now be used seamlessly with the model
- Available in the Model Predictive Control Toolbox
   section of the Simulink Library Browser
- In order to run the nonlinear interior-point solver, the coredata structure returned by nImpcToForces must be provided in the block mask

Block Parameters: Nonlinear MPC (FORCES PRO)		
NonlinearMPC_FORCESPRO (mask) (link)		
The "Nonlinear MPC with FORCES PRO" block lets you simulate with and generate code from a nonlinear MPC controller with FORCES PRO NLP solver.		
Parameters		
"coredata" structure generated from "nImpcToForces" cor	edata	
Block Sample Time		
$\hfill \square$ Use prediction model sample time		
Make block run at a different sample time		
Block Options		
General Online Features		
Additional Inports		
✓ Measured disturbances (md)		
MV targets (mv.target)		
Model parameters (params)		
Additional Outports		
Optimal cost (cost)		
Optimal control sequence (mv.seq)		
Optimal state sequence (x.seq)		
Optimal output sequence (y.seq)		
Optimization status (nlp.status)		
	OK Cancel Help Apply	



# SIMULINK INTEGRATION

- Generated FORCESPRO NLMPC Simulink block can now be used seamlessly with the model
- Available in the Model Predictive Control Toolbox
   section of the Simulink Library Browser
- In order to run the nonlinear interior-point solver, the coredata structure returned by nImpcToForces must be provided in the block mask
- **Simulate** the model and obtain results

% Simulate the model using nonlinear MPC sim('LaneFollowingNMPC')

Block Parameters: Nonlinear MPC (FORCES PRO)		
NonlinearMPC_FORCESPRO (mask) (link)		
The "Nonlinear MPC with FORCES PRO" block lets you simulate with and generate code from a nonlinear MPC controller with FORCES PRO NLP solver.		
Parameters		
"coredata" structure generated from "nlmpcToForces" coredata		
Block Sample Time		
☑ Use prediction model sample time		
Make block run at a different sample time		
Block Options		
General Online Features		
Additional Inports		
MV targets (my target)		
Additional Outports		
Optimal cost (cost)		
Optimal control sequence (mv.seq)		
Optimal state sequence (x.seq)		
Optimal output sequence (y.seq)		
Optimization status (nlp.status)		
OK Cancel Help Apply		



# **CONTROL PERFORMANCE**

#### Longitudinal vehicle velocity is kept close to a driver-set velocity



\_mbotech<sup>\*</sup>

ACC Workshop on Real time NMPC - From Fundamentals to Industrial Applications, June 7, 2022, Copyright (C) Embotech AG

### **CONTROL PERFORMANCE**

#### Lateral deviation e<sub>1</sub> and relative yaw angle e<sub>2</sub> are kept small





ACC Workshop on Real time NMPC - From Fundamentals to Industrial Applications, June 7, 2022, Copyright (C) Embotech AG

#### Generate FORCESPRO nonlinear solver for the targeted platform

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
```





#### • Generate FORCESPRO nonlinear solver for the targeted platform (via options.ForcesTargetPlatform)

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
```





#### Generate FORCESPRO nonlinear solver for the targeted platform (via options.ForcesTargetPlatform)

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
```

```
% Start code generation for Speedgoat x86
mdl = 'LaneFollowingNMPC_Speedgoat_x86';
open_system(mdl); % Open Simulink model
load_system(mdl); % Load Simulink model
rtwbuild(mdl); % Start code generation
```

```
% Deploy application from the start
tg = slrt;
if(~strcmpi(tg.Application, 'loader'))
    tg.unload();
end
tg.load(mdl);
```

```
% Execute application
tg.start();
```





#### Generate FORCESPRO nonlinear solver for the targeted platform (via options.ForcesTargetPlatform)

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
```

```
% Start code generation for Speedgoat x86
mdl = 'LaneFollowingNMPC_Speedgoat_x86';
open_system(mdl); % Open Simulink model
load_system(mdl); % Load Simulink model
rtwbuild(mdl); % Start code generation
```

```
% Deploy application from the start
tg = slrt;
if(~strcmpi(tg.Application, 'loader'))
    tg.unload();
end
tg.load(mdl);
```

```
% Execute application
tg.start();
```



"Embotech: Model based Rapid Prototyping of NMPC"





# embotech.com

